150 ptas.

# mi computer

19

## CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR








# MI COMPUTER CURSO PRACTICO
## DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona, y comercializado en exclusiva por Distribuidora Olimpia, S.A., Barcelona

Volumen II - Fascículo 19

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Asesor técnico: Jesús Nebra

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, A. Cuevas
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, Barcelona-8
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador,* se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.


ISBN: 84-85822-83-8 (fascículo) 84-85822-90-0 (tomo 2)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 238405
Impreso en España - Printed in Spain - Abril 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, Madrid-34.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio Blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Ferrenquín a Cruz de Candelaria, 178, Caracas, y todas sus sucursales en el interior del país.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

**Servicio de suscripciones y atrasados** (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 3371872 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Distribuidora Olimpia (Paseo de Gracia, 88, 5.°, Barcelona-8), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Distribuidora Olimpia, en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

**No se efectúan envíos contra reembolso.**



# Juegos de inteligencia

## Los programas de ajedrez son difíciles de escribir, pero hasta un principiante puede hacer un programa sencillo e "inteligente"



Cortesía de Milton Bradley Ltd

Ian McKinnell

**La mano invisible**
Las máquinas exclusivas para jugar al ajedrez contienen los mismos componentes que los ordenadores personales: una CPU, RAM y el programa en ROM y sólo difieren en el método de entrada y salida. El Phantom, que vemos en la fotografía, utiliza un servomecanismo e imanes que capacitan al ordenador para mover las piezas de ajedrez automáticamente. Cuando, por ejemplo, un caballo salta sobre otra pieza, se emplea un sofisticado algoritmo que quita cualquier obstáculo y que lo vuelve a restituir a su sitio después de efectuada la jugada

Muchas personas, cuando empiezan a escribir sus propios programas para ordenador, sueñan con el día en que sabrán lo suficiente como para poder hacer un programa que juegue al ajedrez. Y esto no se debe, por supuesto, a que no se haya creado software para este juego. Existe gran abundancia de estos programas, tanto en forma de paquetes para ordenadores personales como en forma de máquinas dedicadas exclusivamente a jugarlo. Pero escribir programas de ajedrez se puede llegar a convertir en una obsesión, aun para aquellos programadores que no estén particularmente interesados en él como juego. En este sentido, una posible causa de este interés podría ser el hecho de que nosotros consideramos este juego como un pasatiempo de alto nivel intelectual y, en consecuencia, el ordenador que pueda jugar al ajedrez habrá dado un paso más hacia la creación de una máquina inteligente. Sin embargo, sería muy difícil explicarle a usted, partiendo de cero, cómo escribir un programa completo para jugar al ajedrez. Lo que sí podemos hacer es explicarle algunos de los principios en virtud de los cuales se construyen juegos informatizados "inteligentes", y a un nivel que le permitirá escribir un programa bastante sofisticado en BASIC.

Es importante recordar, no obstante, que los "juegos" a los que estamos aludiendo no son recreativos ni de aventuras ni simulaciones, todos los cuales requieren técnicas de programación diferentes y distinta destreza imaginativa. Comenzaremos nuestro análisis de los juegos de inteligencia con algo que quizá considere un ejemplo trivial, pero que ilustra muchos de los principios que se aplican en la escritura de esta clase de juegos.

La mayoría de los niños conoce el juego tijeras-papel-piedra. Las reglas son sencillas: ambos jugadores piensan en uno de estos tres objetos y después levantan al mismo tiempo una mano en un gesto que representa el objeto escogido. El ganador se decide de acuerdo a estas tres reglas: las tijeras ganan al papel (cortándolo), el papel gana a la piedra (envolviéndola) y la piedra gana a las tijeras (mellándolas).

Para cualquiera que haya seguido nuestro curso de programación BASIC, será muy sencillo escribir un programa que cumpla el papel del ordenador en el juego y lleve el marcador. La función RND se utiliza para seleccionar un elemento de una matriz en serie de tres elementos que contenga 'TIJERAS', 'PAPEL' y 'PIEDRA'. Luego se imprime (PRINT) el elemento elegido al pulsar la barra espaciadora. El

jugador digita su propia elección (el programa confía en su honestidad) y el programa calcula entonces quién ha ganado, visualizando el resultado y acumulando el marcador para él mismo y para su oponente. Si la función RND es verdaderamente al azar, los marcadores deberán ser parejos al cabo de cierta cantidad de vueltas, independientemente de la estrategia que adopte el jugador. Ahora es necesario que determinemos cómo podemos mejorar la estrategia del ordenador para asegurarnos de que él gane al cabo de una serie de vueltas.

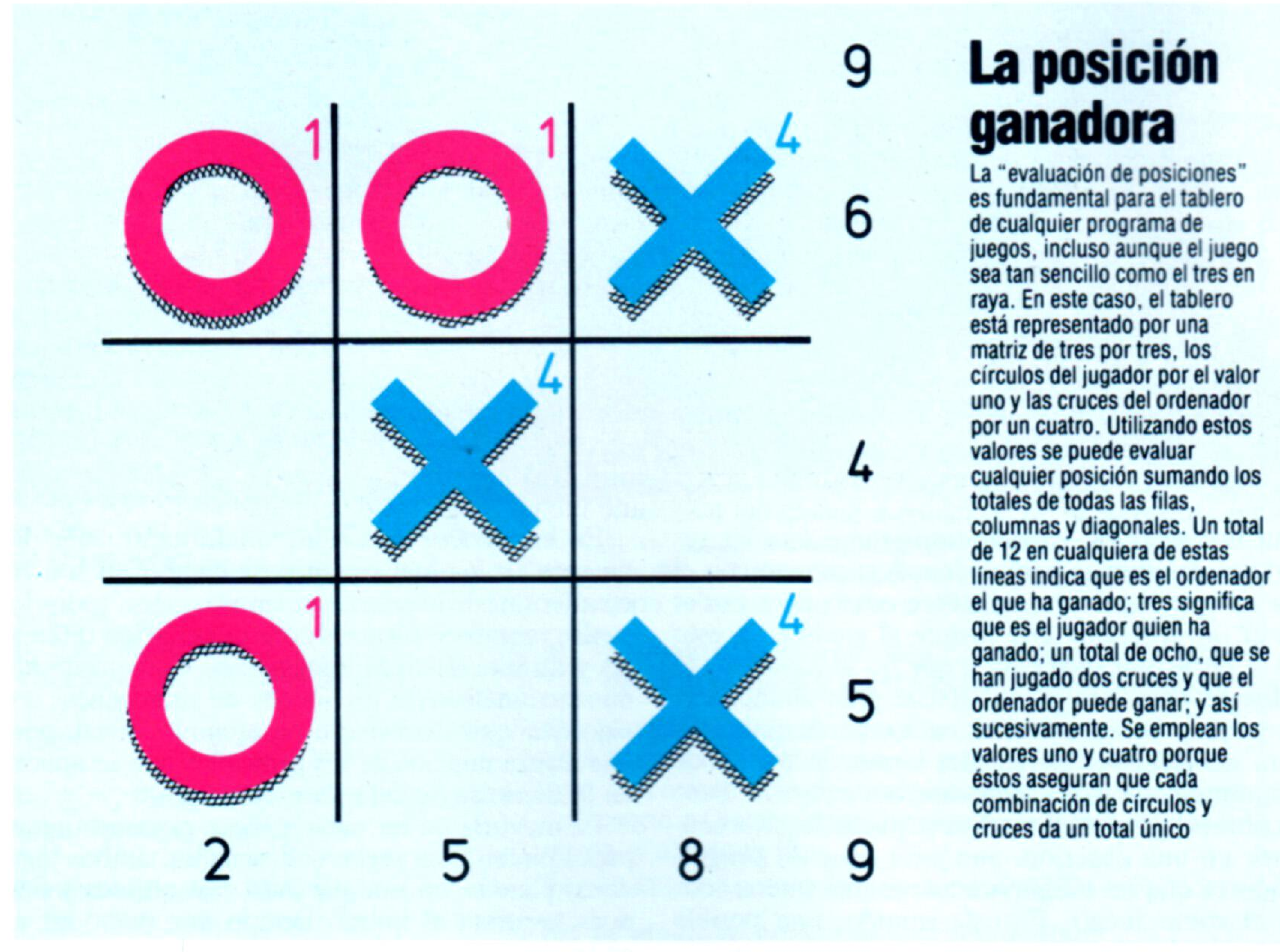
Cuando examinamos las funciones al azar (véase p. 209) aprendimos que generar una secuencia de números verdaderamente aleatorios es una tarea imposible tanto para los humanos como para los ordenadores, aunque estos últimos enfoquen la cuestión mucho mejor. Al cabo de muchas vueltas de nuestro juego, el jugador humano invariablemente favorecerá a uno de los objetos en detrimento de los otros dos. Usted puede escribir una subrutina en su programa que lleve el registro de las elecciones del jugador, utilizando una matriz con tres elementos denominados, pongamos por caso, OPCION(1), OPCION (2) y OPCION(3). Cada vez que el jugador haga una elección, se agrega uno al total del elemento de la matriz correspondiente. El ordenador puede entonces determinar cuál es el objeto que su oponente le presenta más veces y elegir el objeto que venza a la opción favorita del jugador.

Este procedimiento plantea tres problemas. En primer lugar, si el ordenador optara coherentemente por el mismo objeto, el jugador no tardaría mucho en sacar partido de ello. Por consiguiente, por lo general se debe hacer que el ordenador elija entre los tres objetos mediante la función RND, debiéndose agregar una rutina para asegurar que el objeto escogido con mayor frecuencia sea el que derrote a la elección favorita del jugador.

El segundo problema es que el jugador tenderá a cambiar su objeto preferido en el transcurso del juego. De manera que, en vez de llevar el registro de las elecciones del oponente desde el inicio del juego, sería mejor que el programa simplemente registrara, supongamos, las últimas 20 opciones. Ello requeriría una matriz OPCION de 20 por 3 elementos y una subrutina más sofisticada para sumar las tres columnas y predecir cuál sería la mejor elección del ordenador para la siguiente vuelta.

No obstante, el inconveniente más serio de este algoritmo es que el jugador deduzca la estrategia del ordenador. Entonces le resultaría relativamente sencillo jugar de forma que el ordenador perdiera en más de la mitad de las vueltas. El jugador podría, por ejemplo, jugar coherentemente el mismo objeto y después cambiar a otro de forma inesperada, y así una y otra vez. Lo que necesitamos es un algoritmo diferente que evite estos problemas. Sin embargo, valdría la pena desarrollar programas que utilizaran tanto los métodos totalmente al azar como los métodos al azar modificados, y observar los marcadores cuando los emplearan jugadores poco desconfiados.

Dado que los seres humanos son incapaces de tomar una decisión totalmente irracional o al azar, toda elección ha de ser una función de las elecciones anteriores. Esa función puede ser sumamente compleja, sin que el jugador, casi con toda certeza, sea consciente de ello; pero si el ordenador puede calcular una buena aproximación a dicha función, debería ser capaz de ganar la mayor parte de las veces. Dado que cada jugador tendrá su propia fórmula subconsciente, y que probablemente cambiará esa fórmula en el transcurso de un juego largo, se debe hacer que el programa interprete la fórmula mientras se está jugando. Los programas que pueden aprender así se llaman *heurísticos*.




## La posición ganadora

La "evaluación de posiciones" es fundamental para el tablero de cualquier programa de juegos, incluso aunque el juego sea tan sencillo como el tres en raya. En este caso, el tablero está representado por una matriz de tres por tres, los círculos del jugador por el valor uno y las cruces del ordenador por un cuatro. Utilizando estos valores se puede evaluar cualquier posición sumando los totales de todas las filas, columnas y diagonales. Un total de 12 en cualquiera de estas líneas indica que es el ordenador el que ha ganado; tres significa que es el jugador quien ha ganado; un total de ocho, que se han jugado dos cruces y que el ordenador puede ganar; y así sucesivamente. Se emplean los valores uno y cuatro porque éstos aseguran que cada combinación de círculos y cruces da un total único

Liz Dixon

Un programa heurístico permite que el ordenador detecte los cambios en la estrategia del oponente y modifique, por consiguiente, su algoritmo. Dicho programa tendría que llevar un registro de, pongamos por caso, las últimas 50 elecciones de ambos contrincantes, en una matriz. El programa explora constantemente este registro y aplica una técnica estadística conocida como "correlación".

Ésta induce al ordenador a establecer cientos de comparaciones entre la elección del jugador y su elección anterior, o la que precedió a ésta, o la elección que hizo cinco vueltas antes. El ordenador efectúa la misma operación con sus propias elecciones. Consideremos, por ejemplo, la correlación entre la jugada del jugador y su jugada anterior. Denominaremos elemento 1 a las tijeras, elemento 2 al papel y elemento 3 a la piedra. Primero debemos establecer una matriz de tres por tres, a la que llamaremos CORR1, porque representa nuestra primera prueba de correlación. Ahora debemos examinar la historia de nuestro juego, analizando las elecciones del jugador durante las últimas 50 jugadas. Cada vez que a tijeras (1) le siga piedra (3), agregaremos uno al elemento CORR1(1,3); cuando a piedra (3) le siga papel (2), agregaremos uno al elemento CORR1(3,2), y así sucesivamente.

Si el jugador estuviera haciendo sus elecciones auténticamente al azar, entonces en cada elemento de CORR1 debería haber valores aproximadamente iguales; pero es muy poco probable que se dé este caso. Por tanto, si la última vez el jugador eligió papel, entonces el elemento en la fila 2 (papel) de CORR1 con el mayor valor nos proporcionará la mejor predicción acerca de cuál será su próxima elección. Cuanto mayor sea la diferencia entre los elementos de cualquier fila, mayor será la correlación y más fiable será la predicción. Sin embargo, es posible que haya muy poca correlación entre la elección del jugador y su elección anterior, en cuyo caso también deberemos efectuar los cálculos de correlación con la antepenúltima elección, o entre la del jugador y la anterior del ordenador.

Cuando todas las rutinas de correlación predicen resultados distintos para la siguiente jugada del jugador, surge un problema. El programa debe decidir cuál es la sugerencia más fiable. En este juego sencillo, todo lo que hay que hacer es comprobar cuál de las pruebas tiene la correlación más pronunciada. Por ejemplo, la matriz CORR1 podría predecir las siguientes probabilidades: tijeras 51 %, papel 29 %, piedra 20 %; mientras que CORR2 (que, supongamos, compara la elección del jugador con la última elección del ordenador) podría dar: tijeras 24 %, papel 60 %, piedra 16 %. Evidentemente, CORR2 tiene la mejor correlación, de manera que se debería escoger su predicción. De hecho, un programa de juegos inteligente constará de una cierta cantidad de subrutinas, cada una de las cuales trabajará con distintas estrategias y aconsejará a la rutina principal la mejor jugada. La rutina de juego puede considerar a estas subrutinas como si se tratara de un "comité", y actuar en base a una decisión tomada por mayoría. Pero a medida que avanza el juego, puede ponderarse cada rutina según si su consejo ha sido bueno o no.

En el caso de que existiera alguna correlación entre las jugadas o elecciones del jugador y las jugadas anteriores del ordenador, se podría programar alguna clase de factor "de farol" que deliberadamente condujera al jugador a conclusiones erróneas. Donde mejor funciona esto es en los juegos con apuestas en los que éstas aumentan a medida que progresa el juego y es mejor perder las primeras jugadas para ganar las últimas.

```
5 CLS
10 DIM C1(3,3),C2(3,3)C3(3,3)
20 CR=0
30 FOR I=1 TO 3
40 IF C1(PL,I)>CR THEN BG=I:CR=1C1(PL,I)
50 IF C2(PP,I)>CR THEN BG=I:CR=C2(PP,I)
60 IF C3(P3,I)>CR THEN BG=I:CR=C3(P3,I)
70 NEXT I
80 CT=BG-1
90 IF BG=1 THEN CT=3
100 GET PT: IF PT=0 THEN 100
110 REM LA LINEA 100 ESPERA A QUE SE
120 REM PULSE UN DIGITO
130 IF CT=PT-1 THEN CS=CS+1
140 IF CT=PT-2 THEN PS=PS+1
150 IF CT=PT+1 THEN PS=PS+1
160 IF CT=PT+2 THEN CS=CS+1
170 CLS
180 PRINT "SU ELECCION: ";PT
190 PRINT "MI ELECCION: ";CT
200 PRINT "SU MARCADOR ES ";PS
210 PRINT "MI MARCADOR ES ";CS
220 C1(PL,PT)=C1(PL,PT)+1
230 C2(PP,PT)=C2(PP,PT)+1
240 C3(P3,PT)=C3(P3,PT)+1
250 P3=PP
260 PP=PL
270 PL=PT
280 GOTO 20
```

**Un alumno lento**
Este programa, basado en el juego tijeras, papel, piedra, ilustra cómo un programa puede "aprender" a medida que avanza el juego. El ordenador selecciona entre los números 1, 2 y 3, compara su elección con la que usted ha digitado y ajusta el marcador. Se ha utilizado la sentencia GET para que pueda pulsar sencillamente las tres teclas de números en rápida sucesión. Si intenta efectuar su secuencia al azar, descubrirá que tras unas doscientas pulsaciones el marcador del ordenador tomará la delantera. Existe la posibilidad de burlar este programa y, en consecuencia, ganar, pero se pueden agregar rutinas más sofisticadas para impedírselo

En la Universidad del Estado de Nueva York, en Buffalo (según un informe publicado en la revista *Scientific American* en julio de 1978), se hizo jugar entre sí, durante varios miles de partidas a un conjunto de programas de póker. El ganador absoluto fue un programa denominado AEO (*Adaptive Evaluator of Opponents*: evaluador adaptable de oponentes), que elaboraba un juicio inicial acerca del valor de las manos de sus oponentes e iba modificando esta apreciación a medida que avanzaba la partida. El programa SBI (*Sells and Buys Images:* compraventa de imágenes) funcionó sorprendentemente mal: su técnica consistía en establecer faroles con la intención de "venderle" a su oponente una imagen falsa o de "comprar" el estilo de juego de otros. El BP (*Bayesian Player:* jugar a mantener a raya) intentaba hacer deducciones inductivas y mejorar su juego comparando las consecuencias pronosticadas de sus acciones con las consecuencias reales. Por último, el programa AAL (*Adaptative Aspiration Level*: nivel de aspiración adaptable) intentó imitar una característica del juego humano: adaptar el nivel de aspiración (o sea, el grado de riesgo que se está preparado a asumir) a sus mejores resultados anteriores y a su condición presente.

No existen dos programas de ajedrez u otras rutinas artificialmente inteligentes que trabajen de la misma manera. Pero experimentando con las técnicas que hemos esbozado aquí en juegos cada vez más complicados, quizá llegue usted a ingresar en el club de escritores de programas de ajedrez.

# Mapas de memoria

## Los lenguajes de alto nivel como el BASIC administran la memoria automáticamente; de lo contrario, necesitaríamos un trazado de la memoria para hallar el camino a través del ordenador

La CPU, en el corazón del ordenador, posee una escala de direccionamiento que determina el número máximo de posiciones de memoria a las que puede acceder, y en la mayoría de los ordenadores personales éste es de 64 Kbytes. Ese espacio de memoria debe contener toda la RAM y ROM que viene con la máquina, toda ampliación de ambas memorias susceptible de realizarse y todos los chips y puertas para interface, que la CPU también considera como posiciones de memoria. Uno de los aspectos más importantes del diseño de un ordenador es el "mapa de memoria": la lista o el diagrama que especifica qué partes del espacio de memoria se destinan a cada una de las funciones de la máquina. Si su programación está limitada al BASIC, entonces no necesita conocer con detalle el mapa de memoria. Pero si se iniciara en el código de lenguaje máquina o si acariciara la idea de construir sus propios accesorios de hardware, entonces conocer el mapa de memoria sería de vital importancia.

En estas páginas le mostramos el contenido de un mapa de memoria típico. Nuestro ejemplo se aproxima más a un sistema basado en un 6502 que a uno basado en un Z80, pero la mayoría de los elementos son comunes a ambos. Algunos fabricantes imprimen un mapa completo en el manual de instrucciones, mientras que otros mantienen reserva absoluta respecto al diseño. Sin embargo, por lo general es posible hallar usuarios que hayan logrado reconstruirlo a partir de la experiencia.



**Overhead del sistema**
Un ordenador con 4 Kbytes de RAM puede en realidad tener sólo 3 Kbytes disponibles para los programas del usuario. La diferencia estriba en el "overhead" del sistema, una sección de la RAM que queda reservada por el sistema operativo cada vez que se enciende la máquina. Parte de ella se utiliza para las variables del sistema y para los indicadores que señalan en qué parte de la memoria se retienen habitualmente los diferentes datos

**RAM para el usuario**
Las dimensiones de ésta determinan la sofisticación de los programas que usted puede ejecutar. Quizá sea una de las consideraciones más importantes a tener en cuenta a la hora de adquirir un ordenador personal

**Vacío**
En el mapa de memoria se debe reservar espacio para la ampliación de RAM. Algunos sistemas permiten agregar más de 64 Kbytes, pero éstos por lo general son "bancos intercambiables" (un circuito especial cambia la sección correspondiente a RAM dentro y fuera del mapa de memoria en función de las necesidades)

**Pila**
Esta sección reservada de la memoria es para utilización exclusiva de la CPU y está organizada como una estructura de datos LIFO (último en entrar, primero en salir). Un byte se puede "empujar" arriba de la pila o bien se puede "sacar" de arriba para hacerlo volver a la CPU. Cuando se efectúa una rutina GOSUB en BASIC, por ejemplo, la CPU empuja arriba de la pila la posición de la memoria a la cual finalmente habrá de retornar (RETURN). Esta pila se utiliza comúnmente en la evaluación de expresiones aritméticas y en los bucles FOR...NEXT

**Buffers**
En la memoria se debe reservar un buffer para el teclado, para que los caracteres no se pierdan cuando se les dé entrada a mayor velocidad de la que el programa puede procesar. También se requiere un buffer para la cassette, porque la mayoría de los sistemas operativos escriben los datos en ésta en bloques

**Series**
Si el BASIC de su ordenador exige que especifique de antemano la longitud de todas las series, éstas se almacenarán en una tabla de modo similar a las variables dimensionadas. No obstante, de poseer "series dinámicas" cuya longitud pueda variar, entonces los datos verdaderos se almacenarán por separado en una zona de la memoria cuyas dimensiones cambian de modo constante. A intervalos, el sistema operativo promoverá una "recogida de información inservible" que simplemente limpiará la zona de la serie y eliminará los datos que ya estén obsoletos

**RAM del sistema**
En algunos ordenadores la RAM del sistema no está señalada como una parte de la RAM para el usuario. Por lo general se utiliza para la RAM de pantalla (donde un byte corresponde a cada una de las posiciones de caracteres de la pantalla) y para la RAM de color (donde un byte especifica los colores de fondo y de primer plano para cada posición de caracteres). Los ordenadores que posean una gran variedad de modalidades y resoluciones para gráficos necesitarán usar memoria de la RAM para el usuario, lo que produce una ocupación del sistema mucho mayor. En un programa para juegos, por ejemplo, los gráficos pueden representar la mayor parte de las exigencias de la memoria

**Vacío**
Cuando se utiliza un programa de un cartucho, aparece en el mapa de memoria como una ROM de ampliación. Algunas máquinas poseen conectores para ROM libres en el tablero de circuitos impresos para lenguajes adicionales. Éstos también estarán reservados en el mapa de memoria

**Chips de Input/Output**
La CPU se puede comunicar sólo con los dispositivos que aparecen como posiciones en el mapa de memoria, de modo que en el mapa se deben incluir todas las puertas para interfaces y otros chips. Éstos incluyen las interfaces para el teclado, la unidad de cassette, el controlador de video e interfaces externas como la impresora. Por lo general la CPU direcciona la memoria en forma de bloques (casi siempre de 4 Kbytes cada uno). Por consiguiente, los chips de Input/Output podrían ocupar 4 Kbytes del mapa de memoria, aun cuando en realidad sólo se utilice alrededor de una docena de posiciones

**ROM del sistema**
En un ordenador personal se utiliza ROM para almacenar la información que siempre se necesita y que es invariable. El componente fundamental de la ROM es el sistema operativo, que es el conjunto de programas en código de lenguaje máquina de los que depende el funcionamiento del ordenador. Estos programas realizan funciones tales como explorar el teclado y almacenar información en cassette o recuperarla de la misma. Otro componente es el intérprete de BASIC, que traduce los programas a las instrucciones de bajo nivel que la CPU puede comprender

RAM DE PANTALLA 8 K
RAM DE COLOR 1 K
RANURA PARA CARTUCHO DE ROM
CONTROLADOR DE VIDEO 6,5 K
CONTROLADOR DE SONIDO
PIA
GENERADOR DE CARACTERES 4 K
INTERPRETE DE BASIC 2 K
SISTEMA OPERATIVO 12 K
NUCLEO 5 K
1 K
Liz Dixon

**Controlador de video**
Los gráficos más sofisticados, como los sprites y la resolución en modalidades múltiples, se están manipulando cada vez más mediante hardware en lugar de software. El controlador o los controladores de video aparecerán en el mapa de memoria como una docena o cantidad aproximada de registros de un único byte, que determinan todos los componentes visuales, desde el color de fondo de la pantalla hasta la posición exacta de cada sprite

**Controlador de sonido**
Mediante software se pueden conseguir efectos sonoros rudimentarios, pero los ordenadores con voces múltiples, o con control de sonido ADSR, poseen indefectiblemente un controlador de sonido exclusivo, cuya salida alimenta un pequeño amplificador

**PIA**
Los adaptadores de interface periférica *(Peripheral Interface Adaptors)* se utilizan para manipular la mayoría de las interfaces simples para teclado, cassette, palanca de mando e impresora. Los chips más sofisticados (como el 6522 Adaptador de Interface Versátil) pueden convertir datos en paralelo en datos en serie, y viceversa, y poseen cronómetros incorporados que se pueden utilizar para programar o para controlar las velocidades de transmisión

**Generador de caracteres**
Éste es el mejor ejemplo de la utilización de la memoria ROM para almacenar datos en lugar de programas: en este caso los patrones de bits que definen la forma en que los caracteres aparecen en la pantalla. Algunos ordenadores permiten copiar en RAM todo el juego de caracteres o parte del mismo, con lo que se consigue que el usuario pueda definir otros caracteres

**Núcleo**
El "núcleo" (casi todas las máquinas utilizan su propia palabra para designarlo) es el corazón del sistema operativo. Cuando se enciende la máquina, la CPU salta automáticamente a esta posición y comienza a ejecutar este programa. Busca en el área de la RAM para determinar cuánta memoria hay disponible y comprueba si se encuentra enchufado un cartucho de programas. El "núcleo" también manipula las formas de entrada y salida más elementales

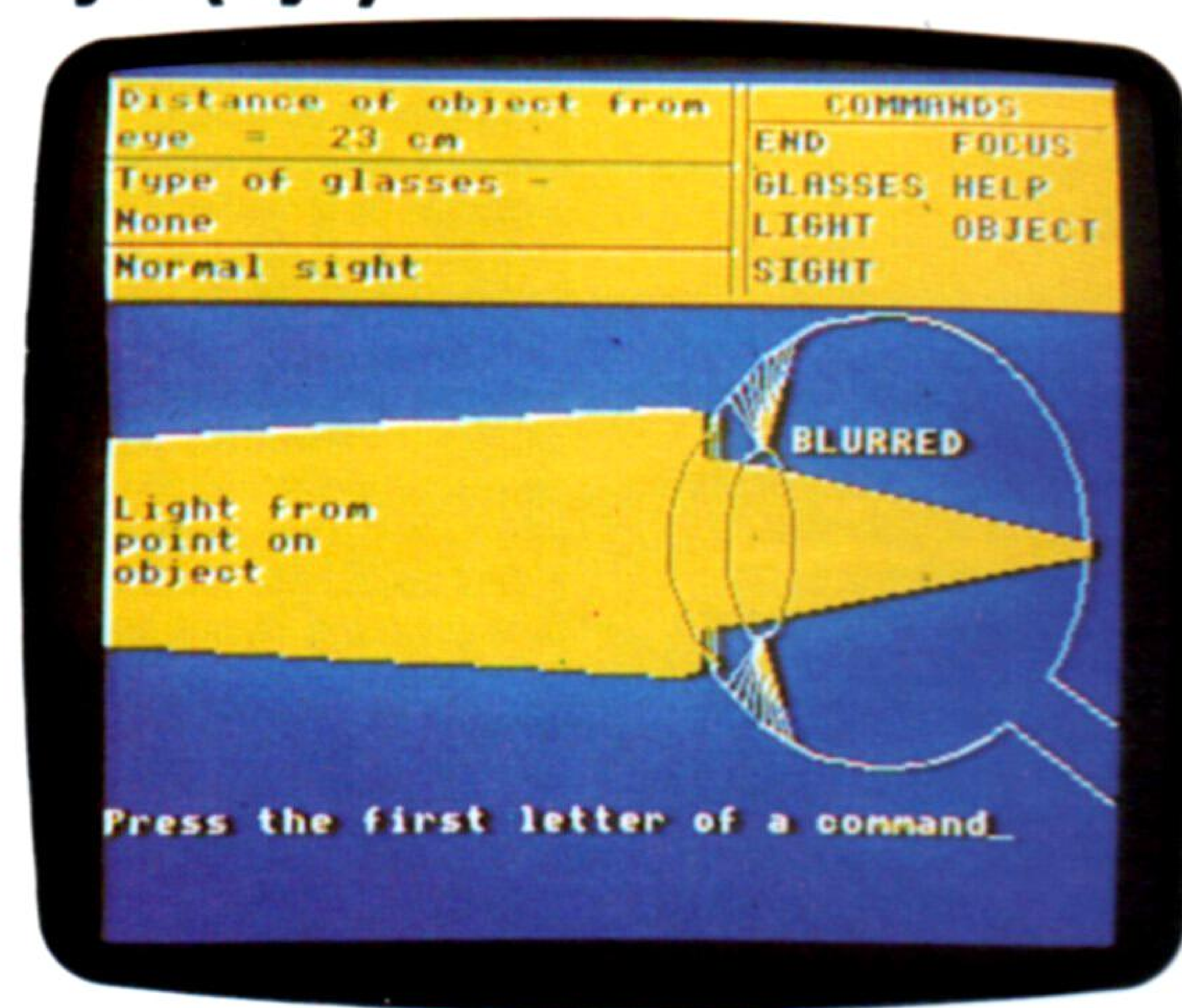
# La simulación

**Este tipo de software permite realizar experimentos sin aparatos ni material, y se puede usar tanto en el hogar como en la escuela**

Los programas de simulación, como los conocidos juegos recreativos que permiten conducir un coche de carreras o pilotar un avión, están diseñados para proporcionar una experiencia lo más parecida posible a la realidad. Existe, no obstante, una amplia gama de software de simulación que intenta no sólo divertir sino también educar. Los programas de este tipo son muy útiles en muchas áreas de los planes de estudio escolares, y especialmente, en aquellas materias (como las ciencias) en las que la experimentación resulta peligrosa, lenta, costosa o complicada.

Los programas de simulación se pueden utilizar como herramientas educativas incluso en el hogar. Por ejemplo, en un programa denominado *Car journey* (Viaje en coche) los niños pueden aplicar sus conocimientos de aritmética y su capacidad de razonamiento para "conducir" un coche. Tal vez los programas de simulación constituyan el más apasionante tipo de software educativo que existe actualmente en el mercado. Lamentablemente, sólo están al alcance de una gama de ordenadores muy reducida: BBC Micro, Spectrum, RML 380Z y Apple, que son las máquinas más usadas en las escuelas.

## Eye (Ojo)




Este programa muestra cómo funciona el ojo y cómo las diversas partes que lo componen han de estar correctamente ajustadas para permitir una visión clara. Imita la trayectoria que siguen los rayos de luz desde un objeto hasta la retina (la parte posterior del ojo donde se forman las imágenes). Usted actúa como si fuera el cerebro, controlando elementos como la distancia del objeto, el tamaño del iris y la longitud focal de las lentes, con el objeto de centrar la imagen en la retina. Se le presenta en la pantalla un diagrama del corte del ojo, con el nombre de cada parte. Utilizando la orden LIGHT se puede trazar la trayectoria de un haz de luz desde un objeto hasta el ojo. Si ha escogido correctamente las otras variables, obtendrá el mensaje IN FOCUS; de lo contrario, aparecerá el mensaje BLURRED (borroso).

Una vez se ha conseguido dominar el funcionamiento de un ojo normal, se pueden simular defectos como la miopía. Aunque originalmente se desarrolló para las clases de física y biología, este programa se suele utilizar en cursos generales de alfabetización informática, como introducción de los jóvenes en el tema de los ordenadores. Este programa de simulación lo fabrica Longmans para el BBC Micro.

## Ballooning (Viaje en globo)




*Ballooning* es un programa educativo para el hogar destinado a los niños de entre ocho y doce años. Lo fabrica Heinemann Educational Software para el Spectrum. El usuario está al mando de un globo aerostático y ha de hacerlo volar. En la pantalla se ve un corte transversal del campo, con el globo posado inicialmente en el suelo. También se muestran cuatro instrumentos: indicador de velocidad de ascenso-descenso, medidor de la temperatura del aire, altímetro e indicador de combustible. Sólo existen dos mandos: un mechero de gas para calentar el aire y hacer que el globo se eleve, y una abertura que deja escapar el aire, haciendo que el globo descienda.

Una sencilla "lección de vuelo" enseña al usuario cómo se utilizan los instrumentos y los mandos para elevarse, volar y hacer aterrizar el globo. Una vez aprendido esto, puede emprender su propia "mi-

sión" de vuelo. Ésta consiste en hacer que el globo aterrice en unos lugares seleccionados (marcados con una X), donde el aeronauta recibe algunas instrucciones. Por ejemplo, una tarea consiste en "ayudar a un granjero a rescatar una oveja": la oveja se encontrará en un campo marcado con una S. Si el globo se quedara sin combustible, debería aterrizar para recoger más bombonas de gas.

Al cabo de algunos intentos infructuosos, en los cuales acabará estrellándose contra los árboles o experimentando otros contratiempos por el estilo, enseguida aprenderá a controlar el globo con gran precisión, valiéndose del mechero. Tampoco le llevará mucho tiempo aprender a leer los instrumentos para poder predecir cuándo debe emplear la abertura o el mechero. Quizá lo más útil sea aprender a controlar un sistema que incorpora un verdadero retardo entre la causa y el efecto.

Este programa es una simulación muy realista, resulta muy entretenido y es de los pocos que agradan por igual a niños y niñas.

## Car journey (Viaje en coche)

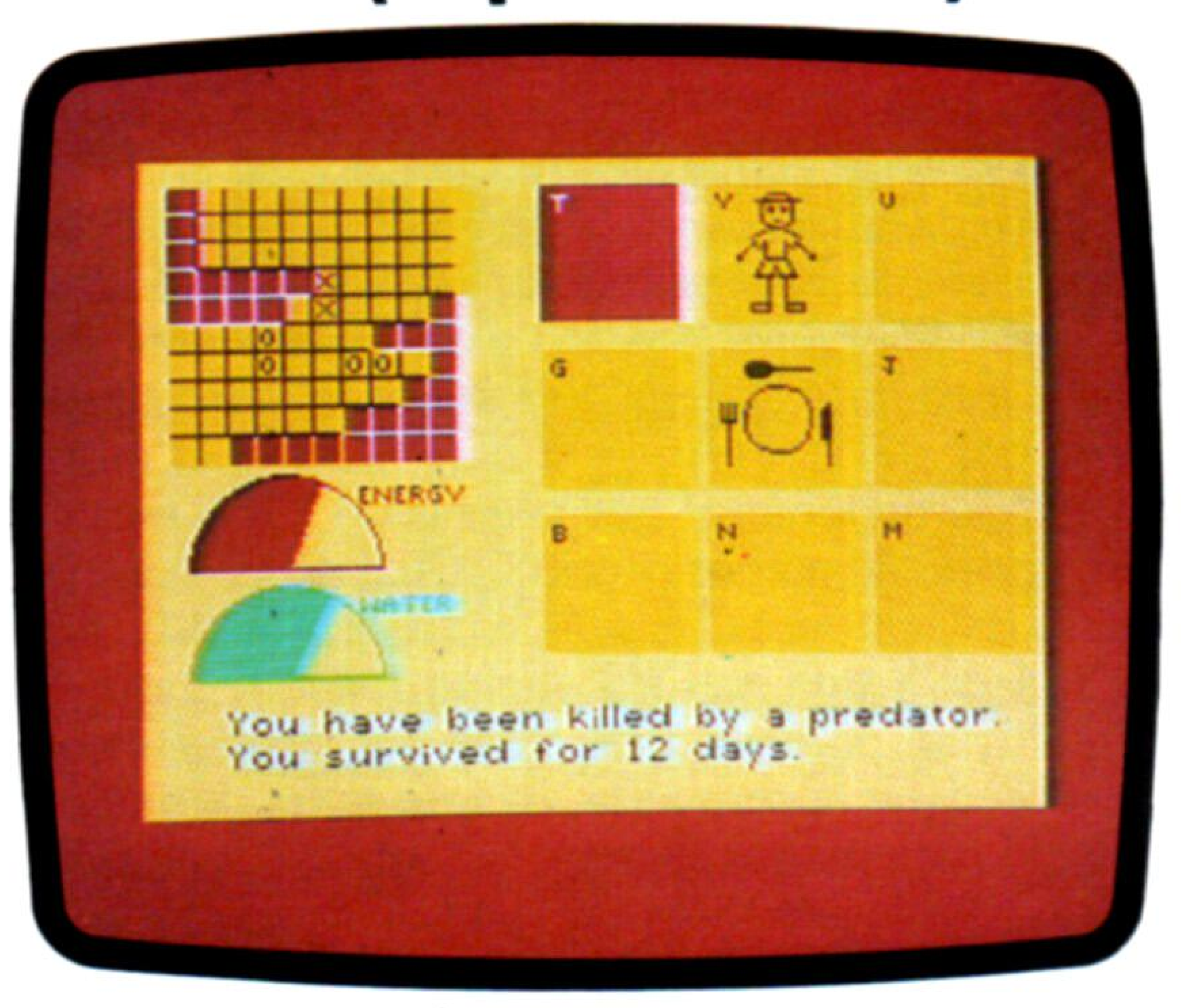



Producido por Heinemann Educational Software para el Spectrum, se trata de un programa educativo para el hogar en el que el usuario asume el papel de propietario de un pequeño servicio de reparto. Se han de tomar diversas decisiones acerca de qué contrato de reparto conviene aceptar, a qué velocidad conducir y qué tipo de vehículo utilizar. Para tomar estas decisiones se deben efectuar cálculos relativos a dinero, distancia, tiempo e incluso consumo de combustible. Se visualiza un mapa de Gran Bretaña que incluye 15 ciudades y las principales carreteras. También se muestran un velocímetro, un cuentakilómetros, un indicador de combustible y un reloj.

La primera tarea consiste en decidir por qué ciudad empezar y después el usuario ha de elegir, entre una lista de doce contratos, el que cree que podrá cumplir. Por ejemplo, un contrato es para recoger unos diamantes en Bristol a las 12 horas y entregarlos antes de las 18 horas del mismo día. Para cumplir este contrato, se debe alquilar un coche, conducirlo hasta Bristol, recoger los diamantes y llevarlos hasta Dover. Si el usuario consigue hacerlo cobra 400 libras, más una bonificación de 10 libras si llega temprano, y entonces podrá escoger otro contrato. Ha de gastar dinero en las paradas donde pernocte, en reparaciones, en gasolina y en multas por exceso de velocidad, y cuando no cumpla un contrato es objeto de una multa de 100 libras. Si acepta una carga pesada, debe descartar el coche y optar por un camión, que es más caro de alquilar, consume más y va más despacio.

Además de desarrollar el conocimiento de los vehículos y las carreteras, *Car journey* también ayuda a desarrollar aptitudes más abstractas, como son la toma de decisiones y el pensamiento lógico. Incluso enseña una teoría económica sencilla, ya que al evaluar los pros y los contras de un determinado contrato, el usuario está efectuando un análisis de coste-beneficio.

## Survival (Supervivencia)




Ian McKinnell

Si alguna vez ha tratado de imaginar lo que sentiría y pensaría si fuera un león (o incluso un ratón), entonces *Survival* es el juego para usted. Le permite interpretar el papel de un animal (halcón, petirrojo, león, ratón, mosca o mariposa) y experimentar algunos de los problemas de su existencia diaria y las decisiones que ha de tomar para seguir vivo.

El mundo se representa en la pantalla mediante una cuadrícula, a través de la cual el usuario se puede mover (su posición se ilustra mediante la letra A) pulsando las teclas. Su principal preocupación consiste en hallar comida (los cuadrados marcados con un 0) y evitar los animales de rapiña (marcados con una X). A medida que se aproxima a un cuadrado marcado, una cuadrícula ampliada situada a la derecha ilustra exactamente con qué animal o con qué alimento se ha encontrado. También se muestran dos medidores que indican cuánta energía y cuánta agua le quedan. Si su nivel energético alcanza un nivel muy bajo, debe encontrar rápidamente algún alimento, y si se le acaba el agua debe acercarse a un cuadrado azul (un río); sin embargo, si accidentalmente "cayera" en un cuadrado azul, se ahogaría.

Algunos animales lo pasan peor que otros: la única fuente de alimentación de la mariposa son las flores, y puede ser difícil encontrar alguna. El halcón, en cambio, puede sobrevivir a base de caracoles, moscas y ratones, pero puede caer víctima de un cazador humano. A través de *Survival* se puede aprender qué lugar ocupan diversas especies en la cadena de alimentación y apreciar algunos de los problemas que plantea la supervivencia en un medio salvaje.

# La mejor solución

**A veces la solución óptima a un problema se halla de manera directa, pero con frecuencia exige la aplicación de matemáticas avanzadas. Los ordenadores asumen esta responsabilidad**

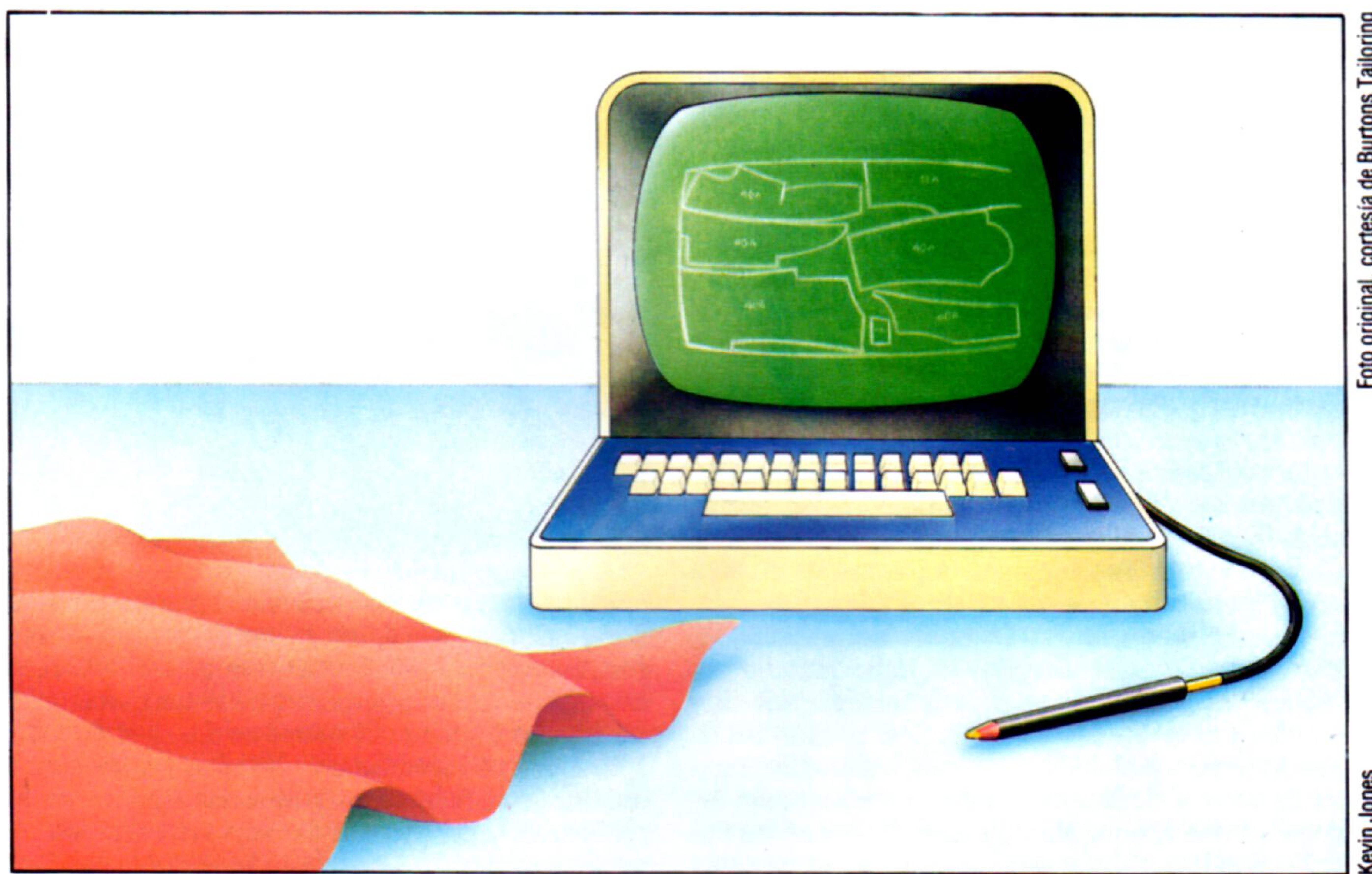
En todas las decisiones que tomamos existe invariablemente una relación de acomodo; por ejemplo, entre costo y eficacia, o entre costo y tiempo. Es improbable que obtengamos un rendimiento máximo absoluto con un costo mínimo absoluto. El resultado "óptimo" estará en algún punto medio entre ambos.

Si tomamos como ejemplo la elección entre dos marcas distintas de detergente, el razonamiento previo a la decisión podría ser el siguiente: «Si compro este detergente, los 150 g me costarán 60 pesetas, y si compro aquella otra marca, pagaré 100 pesetas por 300 g. Pero, ¿y si para obtener el mismo resultado tuviéramos que utilizar un 20 % más del detergente de menor precio? ¿Cuál de las dos marcas sería, entonces, la más barata?" Cuando todo se reduce a una medida común (en este caso, las diferencias porcentuales entre los productos) es fácil predecir la respuesta, incluso antes de realizar cálculo matemático alguno.

El concepto de "sopesar" un cálculo mediante un valor constante es bastante normal y funciona bien cuando las diferencias entre componentes similares (p. ej., el precio o el peso físico) son constantes. Pero cuando estas diferencias cambian en distinta proporción, entonces las operaciones aritméticas se vuelven más complejas y debemos recurrir a otra forma de cálculo (resolviendo una serie de ecuaciones que emplean simultáneamente las mismas variables) con el fin de llegar a la respuesta correcta. Cuando el número de variables sea pequeño, podremos optar por darles entrada en una matriz y manipularla luego. Otra vía consiste en adivinar la respuesta y luego ir modificando sucesivamente la predicción hasta que satisfaga todas las condiciones. Por supuesto, cuanto más acertada sea la predicción, menos tiempo durará el proceso.

Las técnicas de optimización de este tipo son esenciales en el comercio y la industria y se aplican universalmente, en especial en la fabricación y la construcción. La programación lineal, el método del camino crítico y los PERT (técnica de investigación y evaluación de programas) son algunos de los nombres con que se alude a este procedimiento de optimización. En su forma original se adelantó a la era del ordenador en unos 30 años y en la etapa anterior precisaba de un gran aporte de potencial humano para obtener una respuesta correcta en un plazo aceptablemente corto. Las aplicaciones de este tipo son bastante idóneas para los ordenadores personales, pero uno ha de tener presente que la operación por matriz (matriz bidimensional) requiere una gran cantidad de espacio de memoria y que la aritmética de matrices es bastante compleja. Por fortuna, existen varios paquetes de software para sistemas de microordenadores pequeños, de modo que el uso de esta técnica no presenta mayores dificultades.

Una de las áreas comerciales que se ha beneficiado considerablemente de la optimización es la con-



**Ajuste perfecto**
Acomodar los patrones sobre una plancha de material para minimizar el desecho constituye un buen ejemplo de optimización informatizada. Sus aplicaciones van del corte de metales laminados a la sastrería. En esta ilustración, el ordenador visualiza en la pantalla el trazado sugerido, y un operador experimentado puede hacer ajustes menores con la ayuda de un lápiz óptico

Foto original, cortesía de Burtons Tailoring

Kevin Jones

## Optimización de una dieta

| | PRECIO PESETAS | PROTEÍNAS mg | HIDRATOS DE CARBONO mg | GRASA | CALORÍAS | CANTIDAD REQUERIDA |
|---|---|---|---|---|---|---|
| 400 g PATATAS | 20 | 10 | 90 | 0 | 400 | 8,5 kg |
| 200 g SARDINAS | 85 | 80 | 5 | 5 | 500 | — |
| 400 g CARNE | 450 | 150 | 10 | 80 | 1300 | — |
| 1/2 l. LECHE | 30 | 15 | 25 | 20 | 200 | 3,75 l |
| EXIGENCIA MÍNIMA POR SEMANA | | 250 | 1000 | 150 | 10 000 | |

En este ejemplo, el objetivo es hallar la combinación óptima de cuatro productos alimenticios que satisfaga unas exigencias dietéticas mínimas especificadas al mínimo coste. Para ello hemos de decirle al ordenador: los componentes nutritivos (rosa) y el precio por unidad (azul) de cada uno de los productos, además de las exigencias mínimas de cada componente nutritivo en una semana (amarillo). El ordenador descubre el elemento más decisivo y manipula el resto de la cuadrícula en función del mismo para encontrar el equilibrio óptimo, que vemos indicado en verde. En este ejemplo, las exigencias se han satisfecho sólo con patatas y leche con un coste mínimo. (N.B.: esta dieta no es aconsejable)

Kevin Jones

fección de prendas de vestir. El género normalmente viene en unidades de anchura estandarizada (y, algunas veces, también de longitud estandarizada), y el problema del fabricante consiste en minimizar el desecho de material al cortarlo, prestando al mismo tiempo atención a factores como la dirección del pelillo del tejido.

En uno de los talleres de confección más avanzado de Europa, la disposición de los patrones de las piezas sobre un corte de tela para hacer trajes a medida, se calcula utilizando técnicas de optimización, y el resultado sugerido se visualiza en una unidad de representación visual. En este punto, empleando procedimientos de programación orientada a un objeto (véase p. 262), se solicita al operador del ordenador su opinión, basada en sus conocimientos y su experiencia, para mejorar el cálculo del ordenador. Como promedio, el operador introduce mejoras en una de cada cinco ocasiones.

Debido a que los requerimientos de cada tarea, o de cada prenda, son diferentes, se trata de un excelente ejemplo de la utilización inteligente de una optimización informatizada de bajo nivel combinada con la experiencia del operador. Procedimientos de mayor extensión se emplean en industrias que cortan reiteradamente objetos idénticos en un material en planchas, donde se desarrolla todo el proceso de optimización. Dado que la operación de corte o troquel forma parte de una cadena de producción, se repetirá miles de veces. En este caso, el coste del proceso de optimización dividido por el número de unidades fabricadas se cubre con creces con lo ahorrado en material desechable.

El método del camino crítico (MCC), como sugiere su nombre, es un procedimiento para determinar la sucesión de los trabajos más importantes de un proceso de fabricación o construcción, es decir, la parte del trabajo susceptible de retrasar las demás etapas si no se completa según el plan. Su regulación de tiempo es muy estricta: el período requerido para la ejecución de un segmento debe corresponder a su valor en el diagrama o la tabla del MCC. Se utiliza más comúnmente durante la etapa de planificación de los proyectos de construcción, de modo que los constructores puedan destinar hombres y materiales a las diversas fases del proyecto en un orden correcto: fontanería antes de pavimentos, pintores después de yeseros. También en este caso existen paquetes de software para una amplia gama de microordenadores.

Aunque las operaciones aritméticas del proceso de optimización puedan resultar un tanto intimidatorias para quien no posea experiencia, no se puede negar el éxito y la eficacia de esta técnica. Es una de las pocas tareas de procesamiento de números que normalmente se efectúan en microordenadores pequeños y es un componente importante de los sistemas de inteligencia artificial, que, como sucede con frecuencia, reproducen el proceso de aplicación del sentido común.



**Mejores autopistas**
Aparte de los factores sociales, el diseño y el trazado de las autopistas, tanto en las ciudades como en el campo, depende en gran medida de las técnicas de optimización. Lo que más le preocupará al ingeniero será la pendiente de las colinas y el ángulo de las curvas, pero el granjero a quien expropien sus tierras se regirá por unos criterios totalmente distintos. Cuando se planifica una nueva carretera se reúne una inmensa cantidad de datos, que sirven para confeccionar un modelo exhaustivo de la situación. Este modelo se utiliza luego para diversos fines, desde representaciones gráficas hasta la optimización de la ruta

Cortesía del Ministerio de Transporte británico

# Acorn Electron

## En dos años, entre la aparición del BBC Modelo B y el Electron, de Acorn, el desarrollo de la tecnología del microordenador ha sido espectacular

El Acorn Electron es un ordenador elegante que responde a la impresión inicial que transmite: una máquina sólida y bien diseñada. Como versión a escala reducida del BBC Micro, su rendimiento es inferior, pero resulta más cómodo de utilizar. La mayoría de los elementos del BBC Micro se han incorporado al Electron. Por ejemplo, la orden SOUND se utiliza en ambas máquinas en conjunción con la orden ENVELOPE para sintetizar distintos tipos de instrumentos musicales.

El Electron dispone de todas las modalidades de gráficos del BBC Micro, con la excepción del teletexto (MODE 7), que en el BBC se genera mediante un chip especial. El tablero de circuitos del Electron no dispone de este chip, y por ello las visualizaciones al estilo teletexto sólo se pueden producir volviendo a definir la mayoría de los caracteres e imitando el teletexto mediante la utilización de MODE 6 (que, no obstante, está limitada a dos colores). Esto es un lástima, porque en el BBC Micro la modalidad de teletexto es una forma muy económica de producir visualizaciones muy complejas sin necesidad de usar gran cantidad de memoria.

Las facilidades de input y output son, asimismo, menores que en el BBC Micro. La salida visual se realiza a través del canal 36 de TV y por medio de video compuesto y conectores RGB para los monitores monocromáticos o en color. Pero aparte de la conexión para cassette no hay ninguna interface utilizable de inmediato.

La ampliación se puede efectuar a través de un gran conector marginal situado en la parte posterior de la máquina. Lamentablemente, éste sobresale por debajo de un borde de la carcasa, y la única protección existente, en la máquina no ampliada, es una cubierta plástica. En el manual no se proporcionan detalles acerca de las señales que produce, ni ninguna sugerencia relativa a lo que en él se puede conectar. Pero es evidente que aquí es posible enchufar algún tipo de caja de ampliación, porque en las proximidades de la carcasa hay moldeados unos conectores de filamentos de bronce, que se emplean para establecer una conexión mecánica entre el ordenador y el accesorio.

El BASIC incorporado es la ya bien conocida versión del BBC; pero en este caso se ha ampliado considerablemente y posee muchos elementos que

**Cristal del reloj**

**Cristal para control de la señal de TV**
Uno de los factores fundamentales de la estabilidad de imagen del Electron es un cristal separado especial que se utiliza para sincronizar la visualización

**Conector para salida y modulador de TV**

**Conector para video**

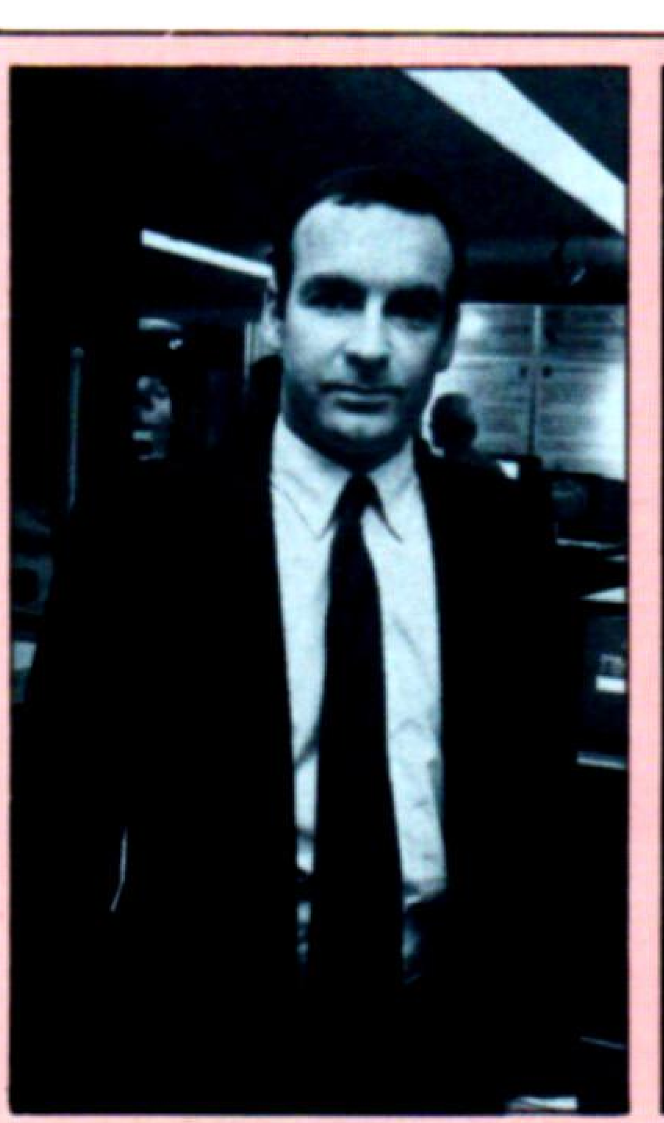
**Conector RGB**

**Conector para cassette**

**Relé para motor de la cassette**
El voltaje que utiliza el motor de una cassette es mayor que el que puede manipular el ordenador, de modo que este relé en miniatura lo aísla de la electrónica del ordenador

**Altavoz**

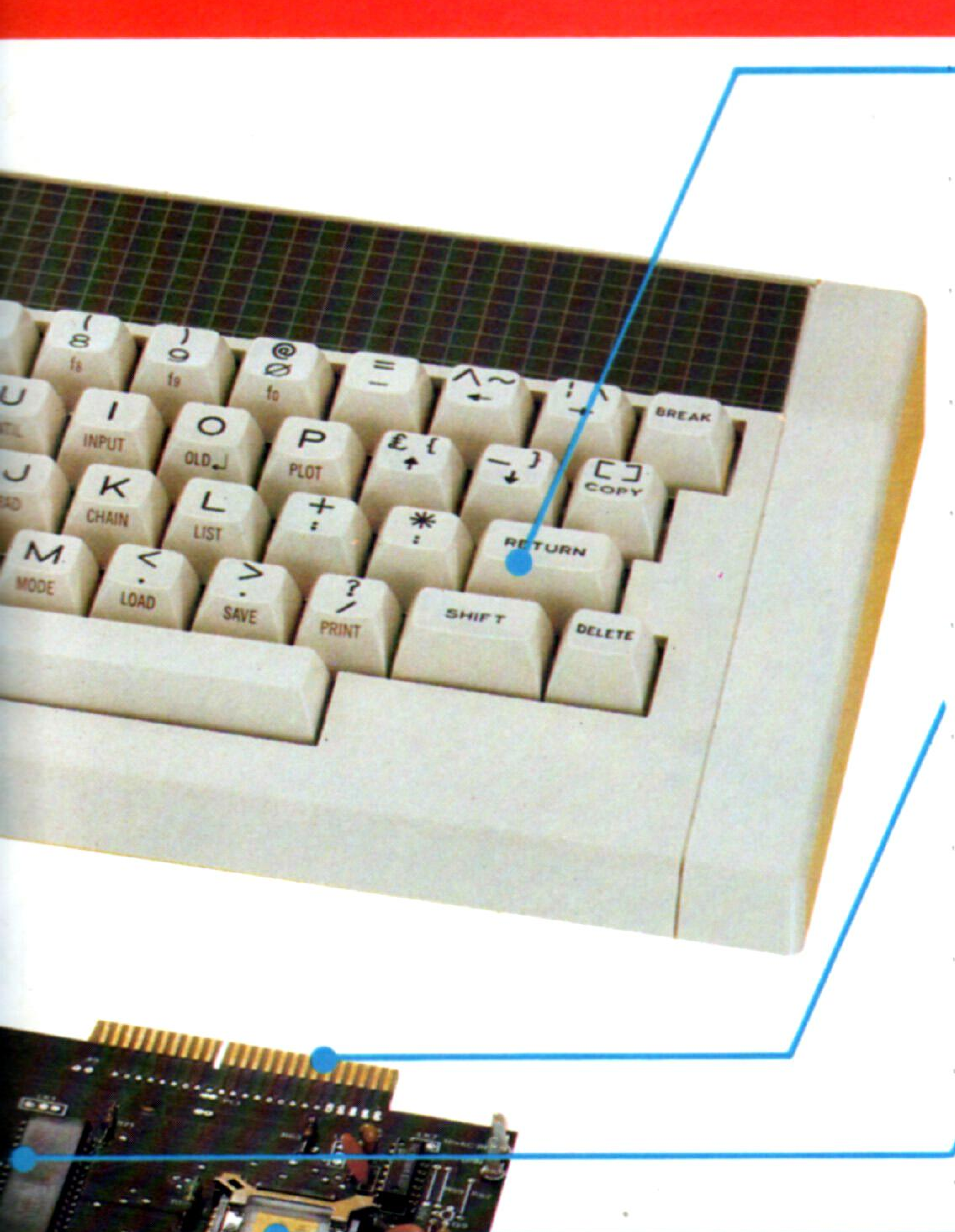
**Conector para teclado**
La cantidad de patillas (22) revela que la salida del teclado no se decodifica en ASCII. Si así fuera, sólo habría 10 patillas a lo sumo (ocho para los datos, más la de 5 v y la de tierra). Esto probablemente está en función de la ULA




**Los creadores del Acorn Electron**
Detrás del Electron está la inteligencia de Chris Curry (izquierda) y Herman Hauser (derecha), quienes fueron asimismo responsables en gran medida del diseño del BBC Micro. Curry, ingeniero de desarrollo, trabajaba para Clive Sinclair cuando éste contrató a Hauser. Luego, Curry y Hauser fundarían Acorn

Judy Goldhill






**Teclado**
Es uno de los mejores teclados de ordenador personal, con verdaderas teclas al estilo de una máquina de escribir y de muy buena calidad. De hecho, es muy similar al del BBC Micro. No hay teclas de función separadas, pero estas facilidades las proporciona la tecla de mayúsculas, que, al ser pulsada junto con una tecla numérica, convierte a ésta en una tecla de función. Lo mismo sucede con las teclas de letras y con tres de las teclas de puntuación, que producen palabras tecla en BASIC si se digitan mientras se mantiene pulsada la tecla de mayúsculas

**Conector para ampliación**
No se proporcionan detalles acerca de los valores de las patillas ni de los tiempos de las señales, pero a través de este conector se dispondrá de la mayor parte del bus del sistema y de las líneas TTL y de potencia. Por consiguiente, la ampliación posible sería considerable

**CPU**
Controlando la máquina hay un procesador 6502A estándar, sincronizado a 1,79 MHz. Éste toma realmente las decisiones, algo que la ULA no puede hacer por sí sola

**ULA**
Esta ULA (*Uncommitted Logic Array:* disposición lógica no comprometida) es la más grande que se ha fabricado hasta hoy. Aparte de la ULA, la CPU 6502, la ROM y la RAM, en el tablero sólo hay otros nueve chips, todos ellos de lógica TTL estándar; cada uno de ellos proporciona tan sólo un puñado de puertas lógicas

**RAM**
Los bytes se cargan de la RAM a la CPU en dos mitades. Primero se accede a los cuatro bits inferiores (un bit proveniente de cada uno de los cuatro chips), y a continuación a los cuatro superiores. En la mayoría de las máquinas, los ocho bits de cada byte se almacenarían en el mismo chip

**Sistema de circuitos acondicionador de potencia**
El Acorn Electron es una máquina inusual por cuanto requiere una fuente de alimentación eléctrica de 19 v. Ésta ofrece la ventaja de ser más estable, pero necesita un circuito más complejo para ser utilizada por el ordenador



**Área de 28 patillas en blanco**
Esta área de 28 patillas, establecida para un chip y con una conexión cercana vacía, sugiere que se podría agregar otra ROM adicional, o bien que se podría emplear un tipo distinto de chip

Chris Stevens

hacen grato el uso de la máquina. De especial utilidad es la rutina OSCLI, que permite que un programa en BASIC le envíe órdenes directamente al sistema operativo, y gracias a ella los usuarios experimentados pueden superar algunas de las limitaciones del BASIC. El paquete ensamblador, que es un elemento exclusivo del BASIC del BBC, se ha ampliado también. Posee palabras tecla adicionales para definir el almacenamiento de variables y la impresión de series, dos trabajos rutinarios en lenguaje Assembler.

El rendimiento del Acorn Electron es superior a la media. La imagen es muy uniforme y nítida, con una buena definición y pureza de color. Cuando disponga de algunas facilidades de ampliación, como unidades de disco, el Electron seguramente se convertirá en una máquina popular.

# ACORN ELECTRON

**DIMENSIONES**
340 × 160 × 65 mm

**CPU**
6502

**VELOCIDAD DEL RELOJ**
1,79 MHz

**MEMORIA**
64 Kbytes de ROM
32 Kbytes de RAM (sin ampliación en el tablero)

**VISUALIZACION EN VIDEO**
Hasta 32 líneas de 80 caracteres. Ocho colores con determinación independiente de fondo y primer plano. 127 caracteres predefinidos y 255 definibles por el usuario

**INTERFACES**
Canal 36 de TV, video compuesto, TTL RGB, cassette, bus del sistema (no documentado)

**LENGUAJE SUMINISTRADO**
BASIC del BBC con ensamblador en línea

**OTROS LENGUAJES DISPONIBLES**
Debería funcionar con otros lenguajes AcornSoft como FORTH y LISP, siempre que se basen en RAM. Los lenguajes basados en ROM, como el BCPL y el PASCAL, son incompatibles con la máquina no ampliada

**VIENE CON**
Manual de instalación y de BASIC, cable de TV, transformador de potencia, cassette de presentación

**TECLADO**
Estilo máquina de escribir, con 56 teclas. Entrada de palabras tecla en BASIC mediante una sola tecla. Diez teclas de función definible por el usuario

**DOCUMENTACION**
Excelente. Hay muchos detalles a disposición del programador interesado y del amante de la experimentación. Todas las palabras tecla en BASIC se explican por separado; hay una sección muy bien realizada acerca del lenguaje Assembler. También están bien descritas las funciones del sistema operativo. Gracias a esta riqueza de información, con esta máquina se puede llevar a cabo la mayoría de las tareas con relativa facilidad

# Impresión a chorro

## Se puede obtener un impreso a todo color gracias a las modernas impresoras que proyectan sobre el papel gotas de tinta de color

Los distintos tipos de mecanismos de impresión que están a disposición del usuario de ordenadores personales producen una impresión de calidad variable. Los mejores resultados se obtienen con las impresoras de impacto de carácter completo (la rueda margarita constituye un ejemplo excelente de este tipo de impresoras); por el contrario, las impresoras electrostáticas y térmicas ofrecen una reproducción más pobre. No obstante, la impresora matricial (véase p. 74), aunque es ruidosa y produce una tipografía de calidad sólo discreta, es el sistema más popular para el uso con ordenadores personales.

Cuando aparecieron por primera vez dispositivos de impresora-plotter como el Tandy CGP 115, se hicieron más evidentes las limitaciones de las impresoras matriciales. Las máquinas impresoras-plotter utilizan lápices esferográficos en miniatura para crear sobre el papel caracteres completos y gráficos lineales, y con frecuencia éstos suelen ser a cuatro colores. Pero las impresoras que más probabilidades tienen de superar en popularidad a la impresora matricial operan sobre el principio de disparar un flujo de gotas microscópicas de tinta sobre una hoja de papel de acuerdo con patrones controlados. Son las *impresoras de chorro de tinta*.

Ya bien establecidas en los sectores industrial y comercial (al igual que la también sofisticada impresora por láser), estas impresoras se están comenzando a introducir en el mercado de ordenadores personales. Funcionan disparando gotas de tinta desde un depósito hasta el extremo de una boquilla muy delgada. Antes de ser expelida, cada pequeña gota de tinta atraviesa un electrodo y recibe una carga eléctrica. El mecanismo de la válvula normalmente es de material piezoeléctrico, lo que permite moldear las gotas mediante vibraciones de muy alta frecuencia.

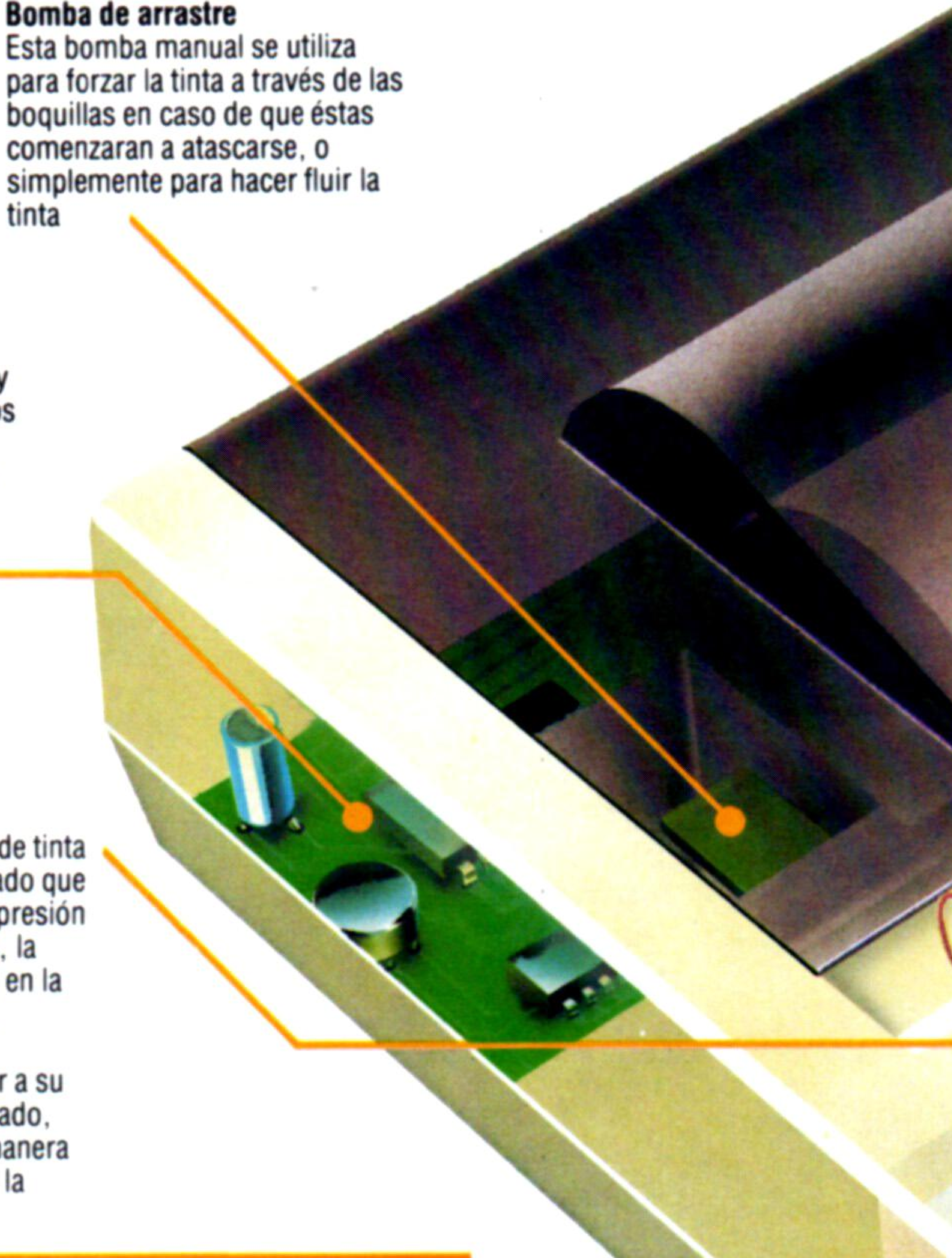
Cuando la gota sale de la boquilla queda suspendida por un campo eléctrico, que también la impulsa hacia el papel. La hoja de papel está extendida sobre una lámina de metal (y no sobre un cilindro de plástico duro o un rodillo de caucho como sucede en una impresora por impacto). La lámina de metal se carga con el potencial opuesto al que retiene la gota y, como las cargas contrarias se atraen, contribuye a llevar la tinta hacia el papel. Esta técnica puede parecer poco fiable pero, sorprendentemente, se producen muy pocas incidencias. Lo peor que puede llegar a ocurrir es que se atasque la boquilla o que las gotas de tinta adquieran un tamaño mayor que el ordinario.

En principio, una impresora de chorro de tinta trabaja, al igual que una impresora matricial, con un solo martillo de impresión. La serie de caracteres ASCII que llega a la impresora se almacena en un buffer hasta que éste está lleno o hasta que reci-

### Misiles dirigidos

Las primeras impresoras de chorro de tinta utilizaban un sistema más sofisticado y eran muy caras. En el interior de la boquilla, un dispositivo piezoeléctrico emitía un flujo constante de gotas de tinta cargadas. Estas gotas se podían guiar verticalmente mediante dos electrodos, mientras la cabeza se movía a través del papel. Cuando no se requería ninguna marca, las gotas se podían conducir a un recipiente y después volverlas a reciclar dentro del depósito principal

PAPEL
ELECTRODOS
DEPÓSITO
BOQUILLA
RECIPIENTE



**Bomba de arrastre**
Esta bomba manual se utiliza para forzar la tinta a través de las boquillas en caso de que éstas comenzaran a atascarse, o simplemente para hacer fluir la tinta

**Tablero de circuitos**
Esta impresora contiene su propio microprocesador 6809 y sus propias ROM y RAM. Todos los datos que entran se deben almacenar en buffer, porque el mecanismo imprime tan sólo una línea de puntos en cada ocasión que la cabeza pasa

**Bloqueo de cabeza de impresión**
El mecanismo de chorro de tinta es muchísimo más delicado que otros mecanismos de impresión y, cuando no está en uso, la cabeza se ha de bloquear en la posición de descanso. El procedimiento operativo inmediatamente posterior a su encendido no es complicado, pero si no se realiza de manera correcta, se puede dañar la máquina

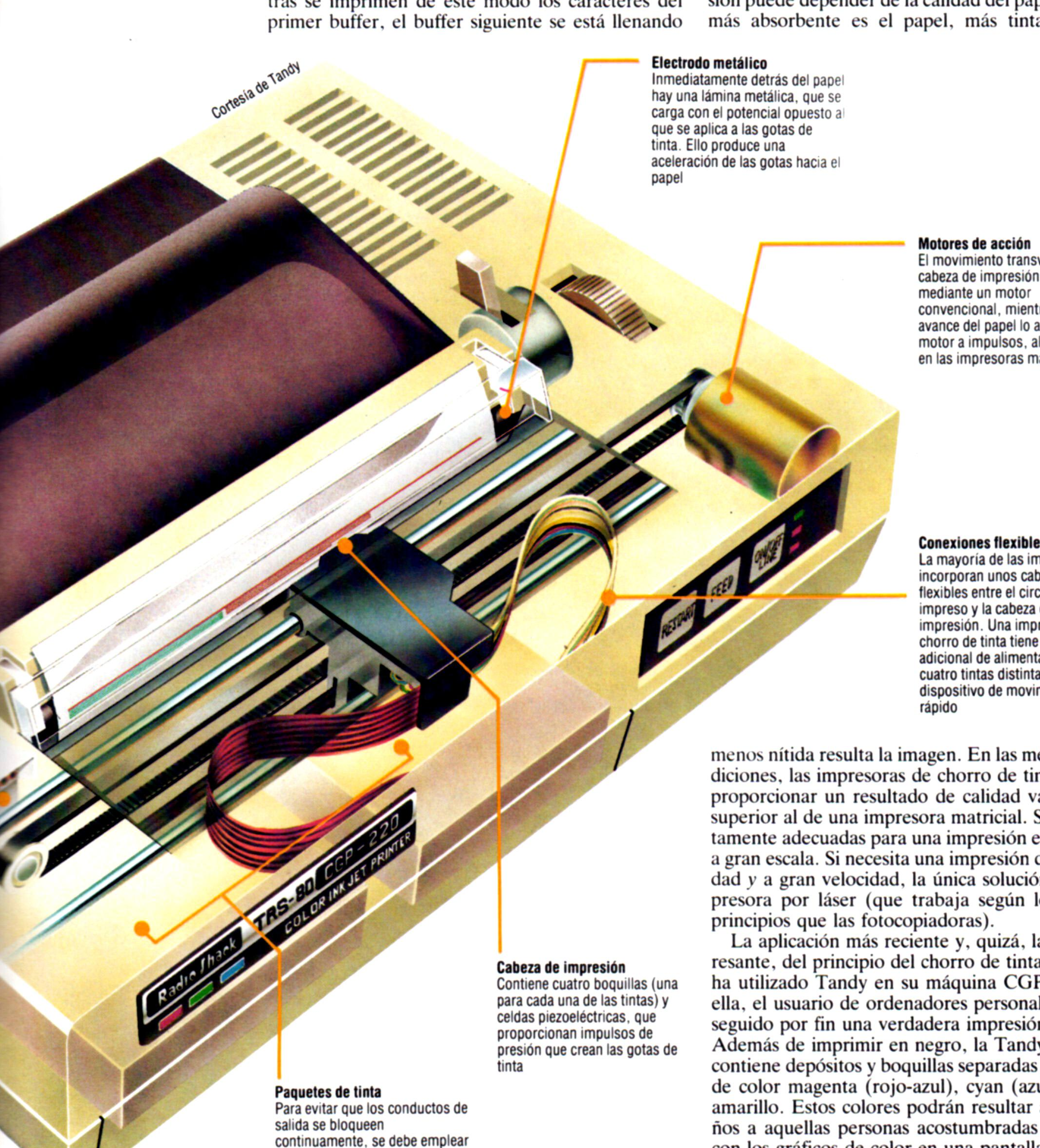
### Caracteres chispeantes

Una variación interesante sobre el tema de las impresoras de chorro de tinta es la impresora de "tinta seca". Disponible como producto de Olivetti y como impresora de Acorn exclusiva para el BBC Micro, la unidad se basa en el principio de la erosión por chispa. Las impresoras de este tipo suelen usar una chispa de alto voltaje para quemar un agujero en un papel plateado especial (un típico ejemplo es la impresora ZX). El sistema Olivetti emplea la chispa para transportar diminutas partículas de carbono desde el extremo de una barra renovable para efectuar una impresión sobre el papel.
La impresora ofrece ciertas ventajas en relación con las impresoras matriciales convencionales: es casi silenciosa, la cabeza de impresión es muy ligera (lo que elimina la necesidad de motores muy potentes) y se puede utilizar prácticamente con cualquier tipo de papel. Los únicos inconvenientes son que la velocidad de impresión es muy baja, la cabeza imprime sólo una línea de puntos cada vez que pasa por el papel y la "tinta" tiende a emborronarse

be un "retorno de carro". La impresora examina entonces uno por uno los caracteres y consulta sus correspondientes patrones en la ROM. Por lo general, cada carácter está compuesto por un conjunto de puntos dispuestos en una cuadrícula de ocho por ocho, y la impresora construye estos patrones sobre el papel. Se requieren ocho pasadas de la cabeza de impresión para crear cada una de las líneas de caracteres, pero el ritmo se acelera, permitiendo que la impresora opere en ambos sentidos. Mientras se imprimen de este modo los caracteres del primer buffer, el buffer siguiente se está llenando para imprimir sus caracteres apenas se vacíe el primero. La única diferencia entre la impresora de chorro de tinta y la matricial de un solo martillo de impresión es que la primera dispara electrónicamente gotas de tinta cargadas contra la página, mientras que en la segunda una aguja imprime a través de una cinta cubierta de tinta.

En su forma comercial, las impresoras de chorro de tinta consiguen producir una hoja impresa en unos segundos. No obstante, la calidad de la impresión puede depender de la calidad del papel: cuanto más absorbente es el papel, más tinta chupa y menos nítida resulta la imagen. En las mejores condiciones, las impresoras de chorro de tinta pueden proporcionar un resultado de calidad varias veces superior al de una impresora matricial. Son perfectamente adecuadas para una impresión empresarial a gran escala. Si necesita una impresión de alta calidad *y* a gran velocidad, la única solución es la impresora por láser (que trabaja según los mismos principios que las fotocopiadoras).

La aplicación más reciente y, quizá, la más interesante, del principio del chorro de tinta, es la que ha utilizado Tandy en su máquina CGP 220. Con ella, el usuario de ordenadores personales ha conseguido por fin una verdadera impresión en color. Además de imprimir en negro, la Tandy CGP 220 contiene depósitos y boquillas separadas para tintas de color magenta (rojo-azul), cyan (azul-verde) y amarillo. Estos colores podrán resultar algo extraños a aquellas personas acostumbradas a trabajar con los gráficos de color en una pantalla de televisión, pero para los pintores son los equivalentes al rojo, verde y azul, y mezclándolos entre sí se puede conseguir todo el espectro.




**Electrodo metálico**
Inmediatamente detrás del papel hay una lámina metálica, que se carga con el potencial opuesto al que se aplica a las gotas de tinta. Ello produce una aceleración de las gotas hacia el papel

**Motores de acción**
El movimiento transversal de la cabeza de impresión se obtiene mediante un motor convencional, mientras que el avance del papel lo acciona un motor a impulsos, al igual que en las impresoras matriciales

**Conexiones flexibles**
La mayoría de las impresoras incorporan unos cables planos flexibles entre el circuito impreso y la cabeza de impresión. Una impresora de chorro de tinta tiene el problema adicional de alimentar con cuatro tintas distintas un dispositivo de movimiento rápido

**Cabeza de impresión**
Contiene cuatro boquillas (una para cada una de las tintas) y celdas piezoeléctricas, que proporcionan impulsos de presión que crean las gotas de tinta

**Paquetes de tinta**
Para evitar que los conductos de salida se bloqueen continuamente, se debe emplear una tinta especial. La tinta negra viene en un paquete separado de la roja, la azul y la amarilla, porque es la que se utiliza con mayor frecuencia

# Prueba sonora

## La síntesis de sonido con el Dragon 32

El Dragon 32 se suministra con un solo oscilador de onda cuadrada para la programación de sonido, pero las órdenes para sonido, extraordinariamente sencillas, de que dispone el BASIC Microsoft Extended Colour permiten la creación de series musicales que con una sola orden ejecutan una melodía aceptable. Lamentablemente, no puede generar ruido. Lo cual resulta sorprendente, ya que es difícil imaginar un juego recreativo que no requiera ruido en algún punto para hacer más interesantes los efectos sonoros.

La orden SOUND sirve sólo para efectos sonoros y el formato es el siguiente:

```
SOUND, P,D
```

donde: P = Tono (1-255) y D = Duración (1-255). El tono es muy impreciso y guarda escasa relación con una escala musical normal, aunque se puede conseguir una aproximación al do central con el valor 89 y el la de diapasón a 440 Hz se halla en 159 más o menos. La duración es igualmente inexacta, pero 16 se aproxima a un segundo, 32 equivale a casi dos segundos y así sucesivamente.

Este programa ilustra cómo se puede utilizar SOUND para un efecto especial; en este caso, con un poco de imaginación, un OVNI despegando:

```
10 FOR P = 10 TO 170 STEP 10
20 FOR D = 16 TO 1 STEP -1
30 SOUND P,D
40 NEXT D
50 NEXT P
```

PLAY puede establecer el tono, la duración y el volumen exactos de una nota. Asimismo, puede especificar una serie de notas a ejecutar (PLAY) con una pausa de tiempo variable entre ellas. Esto hace que resulte muy sencillo crear melodías con diferentes longitudes de notas y silencios, que se ejecutarán (PLAY) con esta única orden:

```
PLAY "T;O;V;L;N;P"
```

donde: T = tiempo (T1-T255); O = octava (O1-O5); V = volumen (V0-V15); L = longitud de nota (L1-L255): N = número de la nota (1-12 o nombre de la nota); y P = pausa antes de la nota siguiente (P0-P255).

No es estrictamente necesario utilizar el punto y coma entre uno y otro parámetro, pero conviene incluirlo para mayor claridad. Este ejemplo es una representación muy arbitraria, porque los parámetros se pueden expresar en cualquier orden. T, O, V y L conservan sus valores hasta que se especifiquen de otra manera. De hecho, T, O, V, L y P corresponden a T2, O2, V15, L4 y P0, respectivamente, si no se especifica en otro sentido, de modo que no hace falta incluirlos siempre en la sentencia PLAY.

Cuando se recurre a una medida del tiempo, como ocurre con L y P, los valores especificados se pueden considerar como si fueran medidas de compás y fracciones: así L1 o P1 es un tiempo entero, L2 o P2 medio tiempo, etc. Los tiempos reales se seleccionan mediante el parámetro T, donde T1 es lento (una nota posee una duración larga) y T255 es muy rápido (una nota posee una duración corta). Además, las longitudes de las notas se pueden definir con más flexibilidad mediante la adición de puntos,

# Pasatiempos luminosos

## Continuamos con las capacidades gráficas del BBC Modelo B

El BASIC del BBC no proporciona la gama completa de órdenes de alta resolución de que disponen algunos microordenadores. Por ejemplo, no posee órdenes CIRCLE ni PAINT. No obstante, la mayoría de las facilidades se pueden imitar utilizando unas pocas líneas del BASIC del BBC.

La pantalla para gráficos posee las mismas coordenadas, sin considerar el nivel de resolución seleccionado, y los ejes se originan en el extremo inferior izquierdo. Las siguientes órdenes permiten controlar la pantalla para gráficos:

```
MOVEx,y
```

Esta orden desplaza el cursor para gráficos hasta el punto de las coordenadas (x,y), pero no traza una línea. Observe que el cursor para gráficos se puede desplazar independientemente del cursor para texto.

```
DRAWx,y
```

Al digitar la orden DRAW se dibuja una línea desde la posición corriente del cursor para gráficos hasta el punto de la pantalla de las coordenadas (x,y).

```
PLOTk,x,y
```

PLOT es una orden muy versátil; su función la gobierna el valor otorgado a la variable k:

| Valor de k | Función |
|---|---|
| 0 | movimiento en relación al último punto |
| 1 | dibujar línea desde origen en color del primer plano |
| 2 | dibujar línea desde origen en color inverso |
| 3 | dibujar línea desde origen en color del fondo |
| 4 | igual que MOVE |
| 5 | igual que DRAW |
| 6 | igual que DRAW pero en este caso en color inverso |
| 7 | igual que DRAW pero en este caso en color del fondo |

como L1... o L5., donde cada punto incrementa la longitud de la nota en la mitad de su valor normal. Por consiguiente, L1... = $1+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}$ = 2 1/2 notas y L5. $=\frac{1}{5}+\frac{1}{10}=\frac{3}{10}$ nota.

No existe ninguna forma categórica de poder representar la relación entre nota y tiempo. Los valores requeridos pueden variar para cada melodía; la mejor forma de seleccionarlos es mediante ensayos y pruebas, que permiten ir eliminando errores. Esto puede ocupar algo de tiempo, pero confiere a la orden una gran flexibilidad.

El parámetro O especifica la octava en la cual se ha de tocar la siguiente nota. O1 comienza con el *do* a 131 Hz y O5 termina con el *si* a 2093 Hz. El $do_3$ (central) empieza la octava O2, que es la octava que se puede omitir. Dentro de una octava, las notas se pueden especificar de dos maneras. En primer lugar, empleando un número para cada una de las doce notas que componen la octava. Así:

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| DO | DO# | RE | RE# | MI | FA |
| 7 | 8 | 9 | 10 | 11 | 12 |
| FA# | SOL | SOL# | LA | LA# | SI |

Esto permite que se pueda especificar una nota como una variable dentro de una octava seleccionada. La segunda manera consiste en emplear directamente el nombre de la nota requerida para que sea más fácil comprender la sentencia en un listado.

Lo que antecede quedará más claro en un ejemplo. La siguiente orden toca fa(6) en la octava omitida O2, durante media unidad temporal (L2) al volumen omitido V15. Luego hace una pausa durante un cuarto de unidad de tiempo (P4), emitiendo después a volumen V20 y durante una unidad temporal L1 la nota *la#* en una octava más alta, la O3. El tiempo se establece en T3:

```
PLAY "T3;L2;6;P4;O3;V20;L1;LA#"
     <  FA  >  <    LA#     >
           intervalo
```

Además, los parámetros T, O, V y L se pueden variar en proporciones preestablecidas en la propia orden mediante la adición de un sufijo:

| Sufijo | Efecto |
|---|---|
| + | Suma uno al valor habitual |
| − | Resta uno al valor habitual |
| > | Multiplica el valor habitual por dos |
| < | Divide el valor habitual por dos |

El formato es: T+, T−, T> o T< para cada parámetro.

La más útil configuración del Dragon es su capacidad para tocar (PLAY) melodías empleando subseries. Éstas primero se definen y luego se tocan (PLAY) en cualquier orden o se repiten:

```
10 A$ = "FA;LA#;SOL"
20 B$ = "DO;RE#;FA;P4;XA$;"
30 PLAY B$
```

Esto define A$ y después lo incluye en B$ como la subserie XA$. La melodía resultante es DO-RE#-FA-P4-FA-LA#-SOL. Esta técnica se puede prolongar en la medida de lo necesario cuando en una pieza musical las secuencias de notas se repiten cierta cantidad de veces. En todos los casos se debe incluir el punto y coma después de la subserie, como en el ejemplo anterior de XA$.

Con números mayores se repiten estas ocho funciones pero con nuevos efectos, como líneas de puntos en lugar de líneas continuas. Los valores de k entre 80 y 87 desempeñan una función particularmente útil. PLOT80,x,y une el punto (x,y) con los dos puntos trazados previamente para formar un triángulo. El triángulo se rellena luego con el color habitual del primer plano. Esto proporciona el único recurso sencillo para pintar (PAINT) formas gráficas.

VDU x es similar a otra orden muy utilizada en BASIC: PRINT CHR$(x). En el artículo anterior sobre los gráficos del BBC Micro vimos que VDU puede ir seguida de una serie de números. VDU v,w,x,y,z equivale a:

```
PRINT CHR$(v);CHR$(w);CHR$(x);CHR$(y);
  CHR$(z)
```

Las órdenes VDU permiten al usuario acceder a aquella parte del sistema operativo del BBC que controla los gráficos y la visualización en pantalla. Si bien estas órdenes se pueden emplear dentro de programas en BASIC, en realidad funcionan independientemente del lenguaje empleado. Por consiguiente, las mismas órdenes VDU se podrían utilizar para una visualización de gráficos en PASCAL o en cualquier otro lenguaje de los que se ofrecen para el BBC. Todas las facilidades para gráficos en BASIC que hemos analizado hasta el momento presente se pueden ejecutar también por medio de la orden VDU adecuada.

Definir caracteres es muy sencillo en el BBC Micro. VDU 23 controla esta función. En capítulos anteriores hemos visto que los códigos ASCII normales se construyen a partir de un bloque de ocho por ocho pixels. Los pixels visibles se pueden representar con un 1 en binario y los no visibles con un 0. Cada fila de ocho bits se puede convertir a su equivalente en decimal, dando un total de ocho números decimales para definir un carácter. VDU 23 permite al usuario volver a definir el carácter con un código ASCII entre 224 y 255. Por ejemplo:

```
10 REM DEFINIR UN CARACTER
20 MODE 2
30 VDU 23,240,16,56,124,146,16,16,16,0
40 PRINT CHR$(240)
50 END
```

Este breve fragmento de programa vuelve a definir el carácter con el código ASCII 240 para crear una forma de flecha. Los últimos ocho números definen esta nueva forma, y la línea 40 imprime (PRINT ) el carácter en la pantalla.

VDU 24 y VDU 28 controlan, respectivamente, la creación de "ventanas" de gráficos y de texto en la pantalla. Empleando estas funciones, la salida de gráficos y de texto a la pantalla se puede limitar a zonas definibles. Esto puede resultar de especial utilidad al diseñar programas interactivos, para los cuales es deseable una pantalla dividida. Lo único que se requiere para definir una ventana de gráficos es especificar las coordenadas para las esquinas inferior izquierda y superior derecha.

**MODE 1**
Este corto listado de programa dibuja en la pantalla una flor de colores en espiral empleando la resolución MODE 1. Observe la utilización de triángulos rellenos para producir los pétalos de la flor

```
10 REM FLOR
20 CLS
30 MODE 1
40 FOR D = 1 TO 3
50 A = 600 : B = 500
60 MOVEA,B
70 FOR C = 1 TO 550 STEP3
80 GCOL0,RND(3)
90 S = (C/(RND(5)+10))
100 X = S*5*SEN(C/16)+A
110 Y = S*5*COS(C/16)+B
120 PLOT85,X,Y
130 NEXT C
140 NEXT D
150 END
```

El patrón en espiral se produce mediante la combinación de seno y coseno en las líneas 100 y 110. Normalmente, esta relación entre las coordenadas *x* e *y* produce un círculo, pero el bucle FOR...NEXT incrementa gradualmente el radio C, produciendo el efecto en espiral. Las coordenadas del centro de la espiral, A y B, se podrían alterar para cambiar la posición de la flor

# Ejecución ficticia

## Para poder utilizar los archivos de datos, primero es necesario crearlos en forma esquemática y luego proveerles de información

Al terminar el capítulo anterior, nos preguntábamos: ¿cómo podemos hacer que un programa lea un archivo que no existe (en cinta o en disco) cuando se ejecuta por primera vez el programa? Con toda probabilidad desearemos que la primera tarea que ejecute el programa sea leer los archivos de datos y asignarles esta información a las matrices o variables. No obstante, si insistimos en escribir primero el archivo, cada vez que se ejecute el programa tendremos que prestar gran atención a la programación para no perder los datos del archivo.

Por fortuna, existe una solución muy sencilla. Muchos paquetes de software comercial incluyen un programa de "instalación" o "preparación" que se debe ejecutar antes de que se pueda utilizar el propio programa, y ésta es la vía que vamos a seguir. Normalmente, dichos programas le conceden al usuario un pequeño margen de "acomodación" (como seleccionar si la impresora a utilizar será una Epson o una Brother, en paralelo o en serie, etc.), pero además crean archivos de datos que con posterioridad serán empleados por el programa principal. Recuerde que, a diferencia de lo que sucede con los archivos de programas, se puede acceder a los archivos de datos a través de cualquier programa (véase p. 316).

Para solucionar nuestro problema y permitir que se pueda llevar a cabo *LEARCH* (la rutina que lee los archivos y les asigna los datos a las matrices), podemos escribir una programación de preparación muy sencilla, simplemente para abrir un archivo y escribir en él un valor ficticio. Elegiremos un valor que el programa formal pueda reconocer como un registro no válido para la agenda de direcciones. Un valor apropiado podría ser la serie de caracteres @VACIO porque es muy poco probable que algún nombre o dirección comience con esta serie particular. *LEARCH* se habrá de modificar ligeramente de modo que cuando abra el archivo y lo lea verifique su valor antes de seguir adelante. Si su ordenador no posee el símbolo @, tendrá que reemplazarlo por '!' u otro carácter, mientras ésta sea una serie que no se produzca naturalmente en su agenda de direcciones. De todos modos, veamos primero el programa de preparación:

```
10 REM ESTE PROGRAMA CREA UN ARCHIVO DE
   DATOS
20 REM PARA QUE LO UTILICE EL PROGRAMA DE
   LA AGENDA DE DIRECCIONES
30 REM ESCRIBE UN REGISTRO FICTICIO QUE
   PUEDA
40 REM SER UTILIZADO POR *LEARCH*
50 REM
60 REM
70 OPEN "0", #1, "DAT.AGCO"
80 PRINT #1,"@VACIO"
90 CLOSE #1
100 END
```

Como ya hemos mencionado antes en nuestro curso de programación BASIC, los detalles relativos a la lectura y escritura de archivos difieren considerablemente de una versión de BASIC a otra, pero el principio es casi siempre el mismo. Primero se debe declarar abierto (OPEN) el archivo antes de que se lo pueda utilizar tanto para entrada como para salida. Luego, la dirección del flujo de datos se declara IN o OUT. A continuación hay que asignarle al archivo un número de "canal". Esto permite tener abierto y en uso más de un archivo a la vez (por el momento, sin embargo, nosotros emplearemos un solo archivo). Por último, se debe declarar el nombre del archivo que deseamos utilizar.

La línea 70 del programa está en BASIC Microsoft y, en principio, es similar a las sentencias OPEN que emplean la mayoría de las versiones de BASIC (el BASIC del BBC es algo distinto; véase p. 319). Por supuesto, OPEN declara que se va a abrir (OPEN) un archivo y "0" que los datos van a salir. #1 es el número que le estamos asignando al archivo para esta operación; de ser necesario, más adelante se podría utilizar un número de archivo diferente. "DAT.AGCO" es el nombre que damos al archivo.

La línea 80 simplemente escribe un único registro en el archivo. La sintaxis de la escritura de datos en un archivo por lo general (en la mayoría de las versiones de BASIC) es exactamente la misma que la sintaxis utilizada para imprimir (PRINT), excepto que la sentencia PRINT debe ir seguida por el número del archivo (#1, en este caso).

La línea 90 cierra (CLOSE) el archivo. Los archivos se pueden dejar abiertos todo el tiempo que necesite el programa; pero los archivos "abiertos" son vulnerables y se deben cerrar lo antes posible para proteger los datos que contienen.

Existe una cierta confusión en cuanto a las formas de utilizar los términos "registro" y "archivo" al aplicarlos a los ordenadores, y esta confusión aumenta cuando hablamos de bases de datos, por un lado, y de archivos de datos, por otro. En una base de datos, el archivo es un grupo completo de información que guarda una relación entre sí. Utilizando la analogía del mueble archivador de una oficina, el archivo podría ser un cajón con el rótulo PERSONAL. Este archivo podría comprender un registro (una ficha dentro de una carpeta) para cada persona de la empresa. Cada registro (ficha) contendría una cantidad de apartados, idénticos para cada registro, con información como NOMBRE, SEXO, EDAD, SUELDO, AÑOS DE SERVICIO, etc.

Un archivo secuencial en disco o en cinta de cassette no se preocupa de cómo el programa utiliza u organiza la información contenida en él. Los archivos de datos tan sólo contienen una serie de ítems de datos y cada uno de ellos constituye un registro. Un solo registro de un archivo de datos, por lo

tanto, normalmente no correspondería a un registro en el sentido en que este término se aplica en la base de datos.

Corresponde al programa leer los registros del archivo de datos y asignarlos a matrices o a variables. Estas matrices y variables necesitan organizarse para conformar un registro "conceptual" que contenga un conjunto limitado de información relacionada.

No existe una relación de punto por punto entre los registros de un archivo de datos y los registros que componen una base de datos.

Una vez se ha ejecutado el programa de preparación, se supone que no se lo volverá a necesitar. De hecho, si se lo volviera a ejecutar, destruiría todos los datos "legítimos" a los que el usuario pudiera haber dado entrada en la base de datos de la agenda de direcciones. Al analizar el programa *LEARCH* modificado veremos por qué sucedería esto.

Cuando se ejecuta el programa, éste no "sabe" si en el archivo de datos hay información legítima o no. Lo primero que hace *LEARCH* es abrir (OPEN) el archivo "DAT.AGCO" y leer el primer registro (o ítem de datos). A éste no se le atribuye un elemento de una matriz, como usted podría imaginar, sino una variable alfanumérica especial que hemos denominado TEST$. Antes de que se lea ningún otro registro, se verifica TEST$ para comprobar si contiene la serie @VACIO. Si TEST$ contiene, en efecto, @VACIO, el programa sabe que en el archivo no hay ningún dato válido y que, por lo tanto, no tiene sentido intentar leer ningún otro dato y asignárselo a las matrices. Por consiguiente, el archivo se puede cerrar y puede continuar el resto del programa. Puesto que en el archivo no hay ningún dato válido, el usuario no puede hacer nada útil hasta que se haya dado entrada al menos a un registro y, por consiguiente, el valor de TEST$ también se puede utilizar para obligar al programa a ir hacia la subrutina *INCREG* a fin de que se agregue al menos un registro válido antes de que se pueda hacer cualquier otra cosa.

Si, por otra parte, el valor de TEST$ no es @VACIO, el programa puede suponer que en el archivo hay algún dato válido y puede comenzar a asignarles los datos a las matrices adecuadas. La subrutina *LEARCH* modificada continúa:

```
1400 REM SUBRUTINA *LEARCH*
1410 OPEN "I", #1,"DAT.AGCO"
1420 INPUT #1, TEST$
1430 IF TEST$ = "@VACIO" THEN GOTO 1530: REM
     CERRAR Y RETORNAR
1440 LET NOMCAM$(1) = TEST$
1450 INPUT #1,MODCAM$(1), CLLCAM$(1), CIUCAM$(1),
     PROCAM$(1), TELCAM$(1)
1460 INPUT #1,INDCAM$(1)
1470 LET TAMA = 2
1480 FOR L = 2 TO 50
1490 INPUT #1,NOMCAM$(L),MODCAM$(L),
     CLLCAM$(L),CIUCAM$(L),PROCAM$(L)
1500 INPUT #1,TELCAM$(L),INDCAM$(L)
1510 REM ESPACIO PARA LLAMAR A LA SUBRUTINA
     "TAMA"
1520 NEXT L
1530 CLOSE #1
1540 RETURN
```

La línea 1420 asigna un único registro del archivo DAT.AGCO a la variable TEST$. Luego la línea siguiente la verifica para comprobar si su valor es @VACIO. Si lo es, se utiliza una sentencia GOTO para saltar a la línea que cierra el archivo (la 1530) y después la subrutina vuelve (RETURN) al programa que la ha llamado. No se efectúan más intentos por leer los datos. Suponiendo que en el archivo no haya ningún dato válido, el control del programa pasará a *INICIL*, que luego llama a *ESBAND*. Todo cuanto esta rutina hace por el momento es establecer el valor de TAMA en 1 si TEST$ = @VACIO. El código para *ESBAND* se proporciona más abajo. Observe que se han escrito varias REM con el fin de dejar espacio disponible para establecer otras banderas en caso de que con posterioridad deseáramos hacerlo.

```
1600 REM *ESBAND*
1610 REM ESTABLECE BANDERAS DESPUES DE *LEARCH*
1620 REM
1630 REM
1640 IF TEST$ = "@VACIO" THEN LET TAMA = 0
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
```

Después *ESBAND* vuelve (RETURN) a *INICIL*, que a su vez retorna al programa principal. *PROPRI* llama luego a *PRESEN*, que visualiza el mensaje de presentación. En el caso de *PRESEN* no es necesario introducir ninguna modificación en la versión que hemos dado previamente para ella.

La siguiente rutina a la cual llama el programa principal es *ELECCN*. Una modificación muy pequeña de la subrutina *ELECCN* de la página 357 establecerá una forma de obligar al usuario a agregar un registro nuevo si el programa se estuviera ejecutando por primera vez.

```
3500 REM SUBRUTINA *ELECCN*
3510 REM
3520 IF TEST$ = "@VACIO" THEN GOSUB 3860
3530 IF TEST$ = "@VACIO" THEN RETURN
3540 REM 'IMMENU'
3550 PRINT CHR$(12)
3560 PRINT "SELECCIONE UNO DE LOS SIGUIENTES"
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. HALLAR REGISTRO (DE NOMBRE)"
3610 PRINT "2. HALLAR NOMBRES (DE NOMBRE
     INCOMPLETO)"
3620 PRINT "3. HALLAR REGISTRO (DE CIUDAD)"
3630 PRINT "4. HALLAR REGISTRO (DE INICIAL)"
3640 PRINT "5. LISTAR TODOS LOS REGISTROS"
3650 PRINT "6. AGREGAR REGISTRO NUEVO"
3660 PRINT "7. MODIFICAR REGISTRO"
3670 PRINT "8. BORRAR REGISTRO"
3680 PRINT "9. SALIR Y GUARDAR"
3690 PRINT
3700 PRINT
3710 REM 'ASOPCN'
3720 REM
3730 LET L = 0
3740 LET I = 0
3750 FOR L = 0 TO 1
3760 PRINT "DE ENTRADA A OPCION (1 - 9)"
3770 FOR I = 1 TO 1
3780 LET A$ = INKEY$
3790 IF A$ = "" THEN I = 0
3800 NEXT I
3810 LET OPCN = VAL(A$)
3820 IF OPCN < 1 THEN L = 0 ELSE L = 1
3830 IF OPCN > 9 THEN L = 0
3840 NEXT L
3850 RETURN
```

Se han agregado dos líneas. La primera comprueba el contenido de TEST$. Esta variable aún contiene el valor que se le atribuye en la rutina *LEARCH*. Si éste es @VACIO sabemos que en el archivo no hay ningún dato válido y, por consiguiente, la única opción adecuada es INCREG, que es el número 6. Si se pasa la prueba, el control pasa a *PRIMERA*, una rutina que visualiza un mensaje apropiado y que

establece la variable OPCN en 6. Cuando la subrutina regresa a la línea 3530, se vuelve a comprobar el contenido de TEST$ (está destinada a pasar la prueba) y la subrutina regresa (RETURN) al programa principal saltándose el resto de la subrutina *ELECCN*, dado que es inadecuada.

Tal vez se pregunte por qué se comprueba TEST$ dos veces. Se hace para impedir que la subrutina retorne (RETURN) a un punto equivocado del programa. Sin la línea 3530, el programa seguiría adelante con el resto de *ELECCN*, presentando el menú de opciones incluso aunque éste no fuera necesario. También evita la utilización de sentencias GOTO, si bien IF TEST$ = "@ VACIO" THEN GOTO 3850 funcionaría igualmente. Las sentencias GOTO hacen que el programa resulte confuso y difícil de seguir (se suele decir que los programas que abusan de sentencias GOTO están en "código spaghetti").

Antes de proseguir analizando *PRIMERA*, volveremos a remitir a los lectores a *LEARCH* y a la sentencia GOTO de la línea 1430. Dado que hemos hecho un razonamiento coherente en contra de la utilización de GOTO, ¿por qué hemos empleado una aquí? Hubiera sido igualmente fácil cerrar (CLOSE) el archivo y retornar (RETURN) simplemente comparando el valor de TEST$ en dos líneas separadas. Aquí hemos preferido utilizar una GOTO para ilustrar uno de los pocos casos en los que su utilización resulta disculpable. Esto es, dentro de un segmento de programa muy breve e identificable, y cuando su función es obvia (y recalcada aún más mediante la observación REM). Las sentencias GOTO nunca se deben emplear para saltar fuera de un bucle (lo cual puede dejar el valor de las variables en un estado impredecible), ni para saltar fuera de una subrutina (lo cual confundiría a la instrucción RETURN) ni para saltar a regiones lejanas del programa.

La subrutina *PRIMERA* es simple y directa: se limpia la pantalla y se visualiza un mensaje que informa al usuario de que se ha de dar entrada a un registro. La línea 3870 establece OPCN en 6, de modo que cuando el control vuelve a pasar a *EJECUT*, la rutina *INCREG* se ejecutará automáticamente. Veamos la codificación para *PRIMERA*:

```
3860 REM SUBRUTINA *PRIMERA* (VISUALIZAR
     MENSAJE)
3870 LET OPCN = 6
3880 PRINT CHR$(12): REM LIMPIAR PANTALLA
3890 PRINT
3900 PRINT TAB(10);"NO HAY REGISTROS EN"
3910 PRINT TAB(7);"EL ARCHIVO. DEBERA EMPEZAR"
3920 PRINT TAB(8);"POR AGREGAR UN REGISTRO"
3930 PRINT
3940 PRINT TAB(0);"(PULSE BARRA ESPACIADORA
     PARA CONTINUAR)"
3950 FOR B = 1 TO 1
3960 IF INKEY$<>" " THEN B = 0
3970 NEXT B
3980 PRINT CHR$(12): REM LIMPIAR PANTALLA
3990 RETURN
```

La subrutina *INCREG*, que aparece en la página 379, incorpora dos modificaciones pequeñas pero importantes respecto a la versión que habíamos ofrecido anteriormente. Una vez se ha dado entrada a los campos como elementos de las diversas matrices, la variable TAMA se incrementa y TEST$ se establece en una serie nula (véanse líneas 10090 y 10100). TAMA es una variable importante que se emplea en diversas partes del programa, con el objeto de que éste pueda saber sobre qué registros se está operando. TAMA se estableció originalmente en 0 como parte de la subrutina *CREMAT*. Posteriormente, en *ESBAND*, se establece en 1 si TEST$ = "@VACIO". Esto se hace para que cuando se ejecute *INCREG* por primera vez, las sentencias INPUT coloquen los datos en el primer elemento de cada matriz. En otras palabras, INPUT "DE ENTRADA AL NOMBRE";NOMCAM$(TAMA) equivale a INPUT "DE ENTRADA AL NOMBRE";NOMCAM$(1).

La línea 10090 incrementa TAMA, de modo que ahora se convierte en 2. Si se volviera a ejecutar *INCREG*, a los datos se les daría entrada en el segundo elemento de cada matriz. Por último, *INCREG* establece a TEST$ en " " en la línea 10100. Esto se hace porque ahora se ha dado entrada a un registro (aunque aún no ha sido almacenado en el archivo de datos de la cinta o disco). Si se volviera a ejecutar *ELECCN*, como se debe hacer para guardar los datos y hacer salir el programa, no querríamos vernos obligados a volver a agregar un registro nuevo. Si no se restaurara TEST$, el programa se metería en un bucle infinito, y la única forma de salir de él sería restaurar o desconectar el ordenador, con lo cual se perderían todos los datos.

Al establecer TEST$ en una serie nula, las condiciones para *ELECCN* en las líneas 3520 y 3530 fracasarán y permitirán que se visualice el menú de opciones. Lo que suceda entonces con TAMA dependerá de qué rutina se ejecute. Hasta ahora sólo hemos asegurado que TAMA = 1 si en el archivo no hay datos válidos, y que éste se incrementa en 1 cada vez que se agrega un registro. Pero ¿qué sucedería si en el archivo hubiera ya registros válidos? Para saberlo, volvamos a analizar *LEARCH*.

La línea 1420 atribuye el primer ítem de datos a TEST$. Si éste no es @VACIO, se da por sentado que es un ítem válido. Los registros del archivo siempre están en el mismo orden, es decir: NOMCAM, MODCAM, CLLCAM, CIUCAM, PROCAM, TELCAM, INDCAM, NOMCAM, MODCAM, etc. Si el primer registro que se lee es un dato válido, debe pertenecer al primer elemento de la matriz NOMCAM$, de manera que la línea 1440 transfiere este dato de TEST$ a NOMCAM$(1). Las dos líneas siguientes completan los primeros elementos de las otras cinco matrices. Ahora sabemos que tenemos un registro (base de datos) completo, y TAMA se establece en 2.

Después, un bucle de 2 a 50 atribuye los registros a las seis matrices, incrementando el índice L a cada ciclo. Ya hemos tomado la decisión de limitar nuestro programa a tratar con archivos de 50 nombres y direcciones, y las sentencias DIM de la subrutina *CREMAT* destinaban espacio para ello. No obstante, cuando comience a utilizar el programa por primera vez, es poco probable que tenga un archivo completo de 50 entradas, de manera que necesitaremos incluir en el programa una rutina que pueda detectar cuándo se produce este caso, establecer la variable TAMA y abortar el bucle de lectura.

Por consiguiente, hemos incluido la línea 1510 para proporcionar una llamada a la subrutina 'TAMA', que desarrollaremos más avanzado el curso. Hay tres formas posibles de abordar este problema. Primero, al escribir los datos en cinta podríamos hacer que el primer registro a escribir fuera la variable TAMA. La subrutina *LEARCH* se podría entonces modificar para que primero leyera TAMA y luego estableciera un bucle de la forma FOR L = 1 TO TAMA para leer los registros. El segundo método, que es preferible (dado que no se contra-

dice con nuestra condición previa para @VACIO de la línea 1430), consiste en establecer un procedimiento a ejecutar después de que se hayan escrito todos los registros, en el cual se pueda escribir al final una bandera especial (tal vez de la forma @PIN). Luego se podría insertar una condición en *LEARCH* para abortar el bucle cuando se encontrara con @PIN.

El tercer método consiste en hacer uso de la función EOF (*End Of File*: fin del archivo) que ofrecen algunos ordenadores personales, que en realidad es una versión automatizada del segundo método. Estas máquinas disponen de una bandera EOF, que normalmente está establecida en 0, es decir, FALSO, pero que asume otro valor (por lo general, 1 para representar VERDADERO) cuando se ha llegado al final del archivo. Algunas versiones de BASIC permiten probar la bandera EOF como una variable en BASIC; en este caso, una construcción de forma:

```
WHILE NOT EOF(N) (N es el número del archivo)
   DO
      INPUT #N (los datos a leer)
ENDWHILE
```

resolverá el problema. En otras máquinas, la bandera EOF se representa como un único bit al que se debe acceder utilizando la sentencia PEEK. Para averiguar si su ordenador dispone de la función EOF, deberá consultar el manual de instrucciones. Dado que esta función difiere mucho de una máquina a otra, no la utilizaremos en nuestro programa. Pero a modo de ejercicio, quizá a los lectores les interese intentar modificar la subrutina *LEARCH* mediante estos tres métodos posibles de tratar con archivos de menos de 50 entradas.

Por lo general, siempre es más fácil escribir programas que traten con archivos de longitud fija, pero abordar el problema de los archivos de "longitud dinámica" en esta etapa temprana nos permitirá modificar más tarde el programa para hacer frente a archivos de más de 50 entradas.

```
4000 REM SUBRUTINA *EJECUT*
4010 REM
4019 IF OPCN = 6 THEN GOSUB 10000: REM VEASE NOTA
     AL PIE DE PAGINA
4020 REM NORMALMENTE 'ON OPCN GOSUB ETC' --VEASE
     NOTA PIE PAGINA
4030 REM
4040 REM  1 ES *ENCREG*
4050 REM  2 ES *ENCNOM*
4060 REM  3 ES *ENCIUD*
4070 REM  4 ES *ENCINI*
4080 REM  5 ES *LISREG*
4090 REM  6 ES *INCREG*
4100 REM  7 ES *MODREG*
4110 REM  8 ES *BORREG*
4120 REM  9 ES *SAPROG*
4130 REM
4140 RETURN
```

La rutina *EJECUT* normalmente no tendría línea 4019 (y por ello el número de línea impar), y la línea 4020 normalmente sería o bien

```
ON OPCN GOSUB número, número, número etc
```

o una serie de:

```
IF OPCN = 1 THEN GOSUB número
IF OPCN = 2 THEN GOSUB número etc
```

Se incluye la línea 4019 para que el programa funcione incluso aunque no se hayan codificado aún las otras subrutinas *EJECUT*

```
10 REM 'PROPRI'
20 REM *INICIL*
30 GOSUB 1000
40 REM *PRESEN*
50 GOSUB 3000
60 REM *ELECCN*
70 GOSUB 3500
80 REM *EJECUT*
90 GOSUB 4000
100 END

1000 REM SUBRUTINA *INICIL*
1010 GOSUB 1100: REM SUBRUTINA *CREMAT* (CREAR MATRICES)
1020 GOSUB 1400: REM SUBRUTINA *LEARCH* (LEER ARCHIVOS)
1030 GOSUB 1600: REM SUBRUTINA *ESBAND* (ESTABLECER BANDERAS)
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN

1100 REM SUBRUTINA *CREMAT* (CREAR MATRICES)
1110 DIM NOMCAM$(50)
1120 DIM MODCAM$(50)
1130 DIM CIUCAM$(50)
1140 DIM PROCAM$(50)
1150 DIM TELCAM$(50)
1160 DIM INDCAM$(50)
1170 REM
1180 REM
1190 REM
1200 REM
1210 LET TAMA = 0
1220 LET RMOD = 0
1230 LET SVED = 0
1240 LET CURS = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN

10000 REM SUBRUTINA *INCREG*
10010 PRINT CHR$(12): REM LIMPIAR PANTALLA
10020 INPUT "DE ENTRADA AL NOMBRE";NOMCAM$(TAMA)
10030 INPUT "DE ENTRADA A LA CALLE";CLLCAM$(TAMA)
10040 INPUT "DE ENTRADA A LA CIUDAD";CIUCAM$(TAMA)
10050 INPUT "DE ENTRADA A LA PROVINCIA";PROCAM$(TAMA)
10060 INPUT "DE ENTRADA AL NUMERO DE TELEFONO";TELCAM$(TAMA)
10070 LET RMOD-1: REM ESTABLECIDA BANDERA 'REGISTRO MODIFICADO'
10080 LET INDCAM$(TAMA) = STR$(TAMA)
10090 LET TAMA = TAMA +1
10100 LET TEST$ = ""
10110 REM INSERTAR AQUI LLAMADA A *MODNOM*
10120 REM
10130 REM
10140 REM
10150 RETURN
```

## Complementos al BASIC




Dado que el Spectrum posee la facilidad de guardar o cargar matrices enteras utilizando la orden SAVE-DATA, como explicamos en la p. 318, la subrutina *LEARCH* será totalmente diferente: se leerán cada una de las matrices (NOMCAM$, CLLCAM$, etc.) sucesivamente. En el próximo capítulo, cuando comencemos a escribir los datos, veremos una versión completa de las subrutinas correspondientes a esta máquina. Mientras tanto, y a modo de ejercicio, los usuarios de ordenadores Spectrum pueden abordar el problema de cómo crear el archivo ficticio que contenga @VACIO, así como determinar cuántas entradas válidas hay en la matriz cuando se lee el archivo




Véase "Complementos al BASIC", p. 175.

OPEN
CLOSE

Véase "Complementos al BASIC", p. 319.

Tony Lodge

P Q R S T U V

# Por partida doble

## Herman Hollerith y James Powers desarrollaron máquinas de tabular. Su rivalidad dominó el mundo de la informática durante seis décadas

James Powers
Las máquinas de Powers eran puramente mecánicas y especializadas en una única aplicación. Así y todo, fue un duro competidor de Hollerith



Cortesía de IBM

Herman Hollerith
Hollerith inventó el lector electromecánico de tarjetas, cuyo desarrollo desembocaría luego en la tabuladora

Las máquinas que Herman Hollerith (véase p. 240) inventara para procesar los resultados del censo de 1890 en Estados Unidos evolucionaron hacia una gama de equipos para el procesamiento de datos con fines generales conocidos como "tabuladoras". Hasta la introducción de los primeros ordenadores comerciales, en la década de los cincuenta, las tabuladoras eran esenciales para el crecimiento de la industria y los negocios. Por ejemplo, durante los años treinta unos importantes almacenes de Pittsburgh experimentaron un sistema de cuentas de clientes en el cual se conectaron 250 terminales distribuidas por el establecimiento a un banco central de tabuladoras mediante las líneas telefónicas. Las mercancías se vendían con unos rótulos perforados y la información se enviaba automáticamente a las tabuladoras, que registraban luego la venta y preparaban una factura para el cliente. Después de comprobarse el margen de crédito del cliente, se enviaba la autorización para la venta a la terminal a través de una máquina de escribir "en línea".

De hecho, la competencia empresarial proporcionó el estímulo inicial para el desarrollo de las tabuladoras. El monopolio que ejercía Hollerith sobre la provisión de equipos para censos se rompió en 1910, cuando el Census Bureau (Departamento de Censos) instó a James Powers a crear máquinas alternativas. Ofreció un sistema de tabuladoras mecánicas que no violaban las patentes de los dispositivos electromecánicos de Hollerith.

La rivalidad entre ellos y las empresas que ambos crearon fomentó el crecimiento de las máquinas para procesamiento de datos.

En 1902 Hollerith diseñó un conmutador de clavijas (parecido al cuadro conmutador telefónico) que podía seleccionar las columnas de la ficha perforada que se habían de agregar y a las que luego había que dar salida. De esta forma, la máquina de Hollerith poseía una capacidad de programación de la que carecían las máquinas de su competidor; Powers siempre fabricó máquinas para aplicaciones específicas; en 1924 patentó una forma de representar datos alfanuméricos en fichas perforadas mediante la utilización de un único agujero en cada columna para un número, y una combinación de agujeros para representar una letra. Hollerith respondió rápidamente con su propio sistema: la nueva tarjeta estándar de 80 columnas. Cada columna de esta tarjeta contenía 12 filas de agujeros que se "leían" mediante escobillas de alambre que completaban un circuito eléctrico con un contacto metálico debajo de la tarjeta.

Las primeras tabuladoras sólo podían contar o acumular totales, pero posteriormente se proporcionaron funciones matemáticas más avanzadas para la manipulación de datos. A diferencia de los ordenadores, que fueron inventados por los científicos con fines matemáticos, la tabuladora se creó para que fuera un procesador de información. Ello estimuló enseguida a elaborar aplicaciones para las nuevas máquinas. Se adaptaron tabuladoras especiales para utilizarlas en tablas de cálculo, en análisis de ondas y en astronomía: en 1930 las tabuladoras identificaron el planeta Plutón. Finalmente, las tabuladoras alcanzaron el nivel de sofisticación suficiente para tratar de forma interactiva grandes cantidades de datos; IBM patentó una que podía llevar los registros de las transacciones de 10 000 cuentas bancarias. Pero su mayor éxito fue cotejar datos a una escala hasta entonces desconocida.

## Las máquinas tabuladoras

En su período de apogeo, a comienzo de la década de los cincuenta, la tabuladora estaba compuesta por ocho unidades separadas. Los datos se introducían en cada tarjeta mediante una "perforadora de tarjetas", que podía procesar 200 tarjetas por hora. Un "verificador" separado comprobaba la exactitud del operador de la perforadora y, cuando las tarjetas se gastaban, una "perforadora de reproducción" creaba copias nuevas. Para dar una referencia sencilla, un "intérprete" imprimía una explicación de los datos en la parte superior de cada columna. La "tabuladora" propiamente dicha acumulaba en las columnas los totales de los datos y transfería la información a una velocidad de 9 000 tarjetas por hora. Esta tabuladora a menudo estaba conectada a una "perforadora de multiplicación", que proporcionaba funciones matemáticas más sofisticadas. El "cotejador" podía comparar los datos de dos grupos de tarjetas o intercalar dos grupos entre sí. Por último, la "clasificadora" podía clasificar un grupo de tarjetas en 13 grupos: uno para cada uno de los 12 agujeros y otro para una columna en blanco.
La operación de la tabuladora se podía modificar mediante códigos de control (en las posiciones 11 y 12) y las tarjetas de control eran de colores brillantes para que pudieran ser reconocidas. Al encontrarse con una tarjeta de control, la tabuladora empezaba una operación nueva, tal como contar un apartado diferente. En el caso de un censo, un ejemplo de apartado sería los datos relativos a una casa, a una calle o a una ciudad. A cada cambio de apartado la tabuladora imprimía un subtotal (en nuestro ejemplo, éste proporcionaría el número de habitantes de una casa, de una calle o de una ciudad)



BBC Hulton Picture Library

# UNION PERFECTA

Así se comportan los periféricos creados por SINCLAIR para SINCLAIR: de forma perfecta. Y es lógico.

Cada vez que SINCLAIR diseña un microordenador, no lo hace de una manera aislada. Simultáneamente crea todos esos periféricos que van a hacer más potente, preciso y útil el microordenador que tiene entre manos.

Periféricos pensados y diseñados para dar un servicio óptimo, pero con un precio razonable, dentro de la filosofía SINCLAIR:

"Hacer la informática accesible a todos".

Por eso cuando creó el ZX 81 vio la necesidad de dotarlo con una ampliación de memoria de 16K RAM para que no quedara pequeño y de una impresora sencilla y barata pero útil y precisa.

Así es la filosofía SINCLAIR. Así son los periféricos de SINCLAIR para SINCLAIR.






# PARA JUGAR A LO GRANDE
# (INSTANTANEAMENTE)

Presentamos **el Interface 2 ZX** Pensado y diseñado por SINCLAIR para unirse a la perfección con tu microordenador Spectrum.

Si a la hora de elegir tu microordenador optaste por el mejor, es lógico que elijas ahora el Interface 2 ZX

Ya habrás podido deleitarte con la más amplia variedad de juegos existentes para tu Spectrum (la más extensa del mercado). Ahora con el Interface 2 ZX vas a tener más ventajas para tu Spectrum:




- Podrás conectar Joysticks para sacarle, aún, mayor rendimiento a tus mejores juegos y divertirte con aquellos exclusivamente disponibles en **Cartuchos ZX:** correr, saltar, volar... a lo grande.
  ¡Menuda diferencia!
- Además, al ser cartuchos con memoria ROM, podrás, con tu SPECTRUM de 16 K, jugar con programas hasta ahora reservados para 48 K, sin ampliar la memoria.
  ¡Vaya ahorro!
- Al conectar el Interface 2 ZX tienes la certeza de poseer un periférico pensado por SINCLAIR para SINCLAIR. Tu microordenador queda a salvo de circuitos poco fiables.
  ¡Un alivio!
- Al adquirir el Interface 2 ZX y los Cartuchos ZX en la red de Concesionarios Autorizados, podrás exigir la tarjeta de garantía INVESTRONICA, única válida en territorio nacional.
  ¡Una tranquilidad!

**Interface 2 ZX y Cartuchos ZX**

Si aún no los tienes
no sabes lo que te pierdes

Solicita una demostración en cualquier Concesionario Autorizado INVESTRONICA.



**DISTRIBUIDOR EXCLUSIVO:**
**INVESTRONICA**

**CENTRAL COMERCIAL:** Tomás Bretón, 60
Tel. 468 03 00 Telex: 23399 IYCO E Madrid.
**DELEGACION CATALUÑA:** Camp, 80 - Barcelona - 22